SNMPv1 Network Management: Organization and Information Models

OBJECTIVES

- *ETF SNMP standard*
  - *History*
  - *RFC, STD, and FYI*
- *Organization model*
  - *2- and 3-tier models*
  - *Manager and agent*
- *Management messages*
- *Structure of management information, SMI*
- *Object type and instance*
- *Scalar and aggregate managed objects*
- *Management information base, MIB*
- *NMS physical and virtual databases*
- *IETF MIB-2 standard*

SNMP management is also referred to as Internet management. We have chosen to call it SNMP management since it has matured to the level that it manages more than the Internet, for example, intranet and telecommunications networks. Any network that uses TCP/IP protocol suite is an ideal candidate for SNMP management. SNMP network management systems (NMSs) can manage even non-TCP/IP network elements through proxy agents.

SNMP management is the most widely used NMS. Most network components that are used in enterprise network systems have network agents built in them that can respond to an SNMP NMS. Thus, if a new component, such as a host, a bridge, or a router that has an SNMP agent built in, is added to a managed network, the NMS can automatically start monitoring the added component. The ease of adding components and configuring them for management has contributed to the acceptance and popularity of the SNMP management system. To quote Marshall Rose [Rose, 1996], who is one of the early architects of SNMP management, the fundamental axiom is, "the impact of adding network management to managed nodes must be minimal, reflecting a lowest common denominator."

SNMP management got started as an interim set of specifications, the ultimate standard being OSI management. Since that did not materialize, SNMP specifications were enhanced by the development of SNMPv2 and SNMPv3. The first version of SNMP is informally referred to as SNMPv1, as it is titled in this chapter. SNMPv2 and SNMPv3 are covered in Chapters 6 and 7, respectively.

We start by giving a real-world example of a managed network in Section 4.1 and show the kind of detailed information one could gather from an NMS. We then learn what SNMP management is and how that enables us to obtain that kind of information. The history of SNMP management goes back to 1970 in managing the Internet. The Internet Engineering Task Force (IETF) has the responsibility to develop Internet standards including network management standards. The standards documents are available free in Request for Comments documents (RFCs). These are covered in Sections 4.2 and 4.3.

The SNMP management model is introduced in Section 4.4 and addresses primarily organization, information, and communication. An NMS comprises management process, agent process, and network elements. We discuss various possible configurations in Section 4.5. There are three messages transmitted by the manager and two by the agent for a total of five messages. Management data are obtained by the manager by polling the agents. Agents respond with requested data. They also generate a few alarms when needed. This simple architecture of SNMP management is described in Section 4.6.

SNMP information model, described in Section 4.7, comprises the Structure of Management Information (SMI) and the Management Information Base (MIB). SMI uses ASN.1 syntax to define managed objects. SMIv1 documented the specifications distinct from the formal ASN.1 definition as it was then expected that OSI would be the future standard. However, that did not happen. Hence, SMIv2 merged the two parts into a concise document. The MIB defines the relationship between managed objects and groups of related objects into MIB modules. MIB-II is a superset of MIB-I and is used in SNMPv1.

SNMP architecture, administration, and access policies, which fall under the communication model, are discussed in the next chapter.

4.1 MANAGED NETWORK: CASE HISTORIES AND EXAMPLES

Let us look at some of the real-world experiences that demonstrate the power of network management before learning how it is accomplished. As with any good technology, the power of technology could result in both positive and negative results. Atomic energy is a great resource, but an atomic bomb is not! An NMS is a powerful tool, but it could also bring your network down, when not "managed" properly.

As part of my experience in establishing a network operations center, as well as in teaching a network management course, we made several visits to see how various corporations and institutions manage their networks. One of the visits was to an AT&T Network Control Center, which monitored the network status of their network in the entire eastern half of the United States. We could see the network of nodes

and links on a very large screen, mostly in green indicating that the network was functioning well. The display screen would automatically refresh every few minutes. We saw nodes or links change color to yellow or red, indicating a minor or major alarm. We would also then see them turn back to green without interference by any human being. What we were seeing was monitoring of a national network from a central monitoring center. Monitoring was done by the NMSs and operations support systems without any human intervention. Even the healing of the network after a failure was accomplished automatically—self-healing network as it is called. Any persistent alarm was pursued by the control center, which tested the network remotely using management tools to isolate and localize the trouble. It was an impressive display of network management capability.

In another visit to a major international news network world headquarters, we were shown the monitoring of not only network failures, but also the performance of networks around the globe. Network Operations Center personnel were able to look at networks in various continents separately, as well as in the global integrated network. The system was putting out not only alarms, but also the cause of the failure, which was accomplished using artificial intelligence built in the system.

On a more intimate level, one of the directors of Information Technology was narrating his experience of resolving a network failure problem using the discovery tool that identified new components in the network. A newly added host interface card was the culprit! This was done from a network operations center, without sending an engineer to the remote site.

There is also another side to the power of this awesome discovery tool, which was an experience of another network manager. He was once asked by one of the departments in the campus to shut off the discovery tool as it was flooding the network and degrading performance. Thus, powerful network management tools also need to be managed to avoid degradation of network performance. There are horror stories of the network coming down when turning on NMSs.

When asked what is the most benefit that he got out of an NMS, one of the managers answered that it is the consistency of administering, for example, configuring the network. This came across as an extremely interesting comment to me as I was once involved in automating the installation and maintenance of a telephone network. One of the operating telephone company managers, who helped in specifications then, commented that what we were trying to accomplish was an impossible task. He said that there were no standard operations procedures for the company that could be automated, and even in one single operations center, no two groups were following the same procedures! Believe it or not, the project was a success.

Let us now illustrate what an NMS could do by monitoring a subnetwork using a commercial NMS. The addresses of network components have been intentionally modified for security reasons.

Figure 4.1 shows a managed LAN that was discovered by an NMS. We show here only a subnetwork of a larger network managed by the NMS. As we mentioned above, an NMS can automatically discover any component in the network as long as the component has a management agent. The management agent could be as simple as a TCP/IP suite that responds to a ping by the NMS. However, agents in the modern network components are more sophisticated. We will study how NMS does an autodiscovery of elements in the network in Chapter 12. Let us accept for now that it has been accomplished somehow.

The managed subnetwork that we are discussing here is an Ethernet LAN that is shown below the backbone cloud in Figure 4.1. It consists of a router and two hubs and is connected to the backbone network. The LAN IP address is 172.16.46.1, and the two hub addresses have been configured as 172.16.46.2 and 172.16.46.3. The LAN IP address, 172.16.46.1, is the address assigned to the interface card in the router. Interface cards in the router and the interface card in each of the hubs are connected by a cat-5 cable forming the Ethernet LAN.

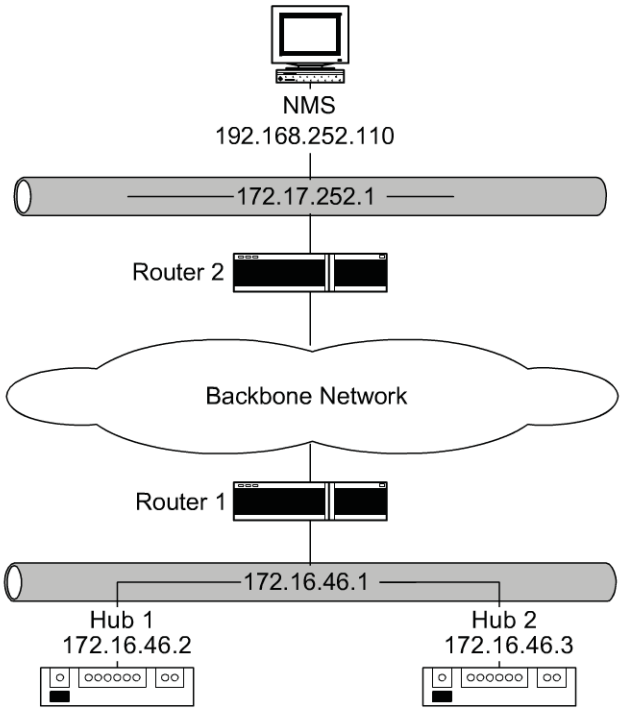


Figure 4.1 Managed LAN Network

The NMS, whose IP address is 192.168.252.110, is physically and logically located remotely from the 172.16.46.1 LAN. It is configured on the LAN 192.168.252.1 and is connected to the backbone network. Information system managers establish conventions to designate a network and a subnetwork. A 0 in the fourth decimal position of an IP address designates a network, and a subnetwork is designated with a 1 in the fourth position of the dotted decimal notation. Thus, 172.16.46.1 is a LAN subnetwork in the network 172.16.46.0.

Once network components have been discovered and mapped by the NMS, we can query and acquire information on system parameters and statistics on the network elements. Figure 4.2 presents system information on three network elements in the managed LAN gathered by the NMS sending specific queries asking for system parameters.

Figure 4.2(a) shows that the network element is designated by 172.16.46.2. No specific title or name has been assigned to it. System description indicates that it is a hub made by a 3Com vendor, with its model and software version. It also gives the system object ID and how long the system has been up without failure. The format of the System ID follows the format shown in Figure 3.10 with the 3Com node under enterprises node being 43. The last three node numbers, 1.8.5, following 43 describe the private MIB of 3Com. The *System Up Time* indicates that the system has been operating without failure for over 286 days. The number in parenthesis is in units of one hundredths of a second. Thus, the hub designated by the IP address 172.16.46.2 has been up for 2,475,380,437 hundredths of seconds, or for 286 days, 12 hours, 3 minutes, 24.37 seconds. System Description and System Object ID are factory set and the rest are user settable.

Figure 4.2(b) shows similar parameters for the second hub, 172.16.46.3, on the LAN. Figure 4.2(c) presents system information sent by the router on the network to the NMS's queries. The system name for the router has been configured and hence the query received the response of the name, router1. gatech.edu.

```
Title:  System Information      : 172.16.46.2
Name or IP Address:  172.16.46.2

System Name             :
System Description      : 3Com LinkBuilder FMS, SW version:3.02
System Contact          :
System Location         :
System Object ID        : .iso.org.dod.internet.private.enterprises.43.1.8.5
System Up Time          : (2475380 437) 286 days, 12:03:24.37
```

(a) System Information on 172.16.46.2 Hub

```
Title:  System Information:  172.16.46.3
Name or IP Address:  172.16.46.3

System Name             :
System Description      : 3Com LinkBuilder FMS,  SW version:3.12
System Contact          :
System Location         :
System Object ID        : .iso.org.dod.internet.private.enterprises.43.1.8.5
System Up Time          : (3146735182) 364 days, 4:55:51.82
```

(b) System Information on 172.16.46.3 Hub

```
Title:  System Information: router1.gatech.edu
Name or IP Address:  172.16.252.1

System Name             : router1.gatech.edu
System Description      : Cisco Internetwork Operating System Software
                        : IOS (tm) 7000  Software (C7000 -JS-M), Version
                        : 11.2(6),RELEASE SOFTWARE (ge1)
                        : Copyright (c) 1986 -1997 by Cisco Systems, Inc.
                        : Compiled  Tue 06 -May -97 19:11 by kuong
System Contact
System Location         :
System Object ID        : iso.org.dod.internet.private.enterprises.cisco.ciscoProducts.
                          cisco 7000
System Up Time          : (315131795) 36 days, 11:21:57.95
```

(c) System Information on Router

Figure 4.2 System Information Acquired by an NMS

Figures 4.3(a), (b), and (c) present the data acquired by the NMS from the interface cards on the two hubs and the router, which are on LAN 172.16.46.1. They are addresses associated with each interface. At the top of each figure are the titles and IP address or name of the network interface card used by the NMS to access the network component. Thus, in Figure 4.3(a), the title and the name or IP address are 172.16.46.2. Note that the IP address 172.16.46.3 is the address as seen by the NMS traversing the router. In Figure 4.3(b), the IP address 172.16.46.3 is the access address of the second hub on the

Title: Addresses: 172.16.46.2
Name or IP Address: 172.16.46.2

Index	Interface	IP Address	Network Mask	Network Address	Link Address
1	3Com	172.16.46.2	255.255.255.0	172.16 46.0	0x08004E07C25C
2	3Com	192.168.101.1	255.255.255.0	192.168.101.0	<none>

(a) Addresses on 172.16.46.2 Hub Ports

Title: Addresses: 172.16.46.3
Name or IP Address: 172.16.46.3

Index	Interface	IP Address	Network Mask	Network Address	Link Address
1	3Com	172.16.46.3	255.255.255.0	172.16 46.0	0x08004E0919D4
2	3Com	192.168.101.1	255.255.255.0	192.168.101.0	<none>

(b) Addresses on 172.16.46.3 Hub Ports

Title: System Information: router1.gatech.edu
Name or IP Address: 172.16.252.1

Index	Interface	IP Address	Network Mask	Network Address	Link Address
23	LEC.1.0	192.168.3.1	255.255.255.0	192.168.3.0	0x00000C3920B4
25	LEC.3.9	192.168.252.1	255.255.255.0	192.168.252.0	0x00000C3920B4
13	Ethernet2/0	172.16.46.1	255.255.255.0	172.16.46.0	0x00000C3920AC
16	Ethernet2/3	172.16.49.1	255.255.255.0	172.16.49.0	0x00000C3920AF
17	Ethernet2/4	172.16.52.1	255.255.255.0	172.16.52.0	0x00000C3920B0
9	Ethernet1/2	172.16.55.1	255.255.255.0	172.16.55.0	0x00000C3920A6
2	Ethernet 0/1	172.16.56.1	255.255.255.0	172.16.56.0	0x00000C39209D
15	Ethernet2/2	172.16.57.1	255.255.255.0	172.16.57.0	0x00000C3920AE
8	Ethernet1/1	172.16.58.1	255.255.255.0	172.16.58.0	0x00000C3920A5
14	Ethernet2/1	172.16.60.1	255.255.255.0	172.16.60.0	0x00000C3920AD

(c) Addresses on Router Ports (Partial List)

Figure 4.3 Addresses Information Acquired by an SNMP NMS

172.16.46.1 LAN. Figure 4.3(c) shows the title and name or IP address as router1.gatch.edu. By using a network lookup command, the IP address of router1.gatech.edu can be recognized as 172.16.252.1. This is the backbone interface address of the router and is the interface on the router as seen by the NMS traversing the backbone network.

In Figures 4.3(a), (b), and (c), we notice that there are six columns of data. The first column is the index, which identifies the row in the matrix. Each row is a collection of various addresses associated with an interface. The second column describes the port id. For example, hubs 1 and 2 have 3Com cards in them. Column 2 of Figure 4.3(c) identifies the card and the port of the interface. For example, the

row with index 2 identifies Ethernet 0 card/port 1. The IP address of the interface card is presented in the third column of the matrix. The IP address in the third column and the network mask address in the fourth column are "and-ed" in modula-2 arithmetic to obtain the network address presented in the fifth column. This implies that all packets destined for network address 172.16.46.0 will be accepted by hub 1. The sixth and the last column in Figure 4.3, the link address, contains the MAC address. In the first row of Figure 4.3(a), 08004E07C25C is the MAC address of the hub 1 interface card. Link addresses in the second rows of Figures 4.3(a) and (b) are presented as "none," as they are non-LAN interfaces.

The Figure 4.3(c) matrix has many rows, as it is a router with many interface cards, each with multiple ports. For example, each Ethernet card has four physical ports. LEC 1.0 and LEC 3.9 are ATM LAN Emulation Card interfaces.

4.2 HISTORY OF SNMP MANAGEMENT

SNMP management began in the 1970s. Internet Control Message Protocol (ICMP) was developed to manage Advanced Research Project Agency NETwork (ARPANET). It is a mechanism to transfer control messages between nodes. A popular example of this is Packet Internet Groper (PING), which is part of the TCP/IP suite now. We learned to use this in the exercises in Chapter 1. PING is a very simple tool that is used to investigate the health of a node and the robustness of communication with it from the source node. It started as an early form of network-monitoring tool.

ARPANET, which started in 1969, developed into the Internet in the 1980s with the advent of UNIX and the popularization of client–server architecture. Data were transmitted in packet form using routers and gateways. TCP/IP-based networks grew rapidly, mostly in defense and academic communities and in small entrepreneurial companies taking advantage of the electronic medium for information exchange. National Science Foundation officially dropped the name ARPANET in 1984 and adopted the name Internet. Note that the Internet is spelled with a capital I and is limited to a TCP/IP-based network. An Internet Advisory Board (IAB) was formed to administer Internet activities, which are covered in the next section.

With the growth of the Internet, it became essential to have the capability to remotely monitor and configure gateways. Simple Gateway Monitoring Protocol (SGMP) was developed for this purpose as an interim solution. The IAB recommended the development of SNMP that is a further enhancement of SGMP. Even SNMP management was intended to be another interim solution, with the long-term solution being migration to the OSI standard CMIP/CMIS. However, due to the enormous simplicity of SNMP and its extensive implementation, it has become the de facto standard. SNMPv2 was developed to make it independent of the OSI standard, as well as adding more features. SNMPv2 has only partially overcome some of the limitations of SNMP. The final version of SNMPv2 was released without one of its major enhancements on its security feature due to strong differences in opinion. SNMPv3 addresses the security feature.

4.3 INTERNET ORGANIZATIONS AND STANDARDS

4.3.1 Organizations

We mentioned in the previous section that the IAB recommended the development of SNMP. The IAB was founded in 1983 informally by researchers working on TCP/IP networks. Its name was formally

changed from the Internet Advisory Board to the Internet Architecture Board in 1989 and was designated with the responsibility to manage two task forces—the Internet Engineering Task Force (IETF) and the Internet Research Task Force (IRTF).

The IRTF is tasked to consider long-term research problems in the Internet. It creates focused, long-term, and small research groups working on topics related to Internet protocols, applications, architecture, and technology.

With the growth of the Internet, the IETF organization has grown to be the protocol engineering, development, and standardization arm of the IAB.

The Internet Network Information Center (InterNIC) is an organization that maintains several archives that contain documents related to the Internet and the IETF activities. They include among other documents, Request for Comments document (RFC), Standard RFC (STD), and For Your Information RFC (FYI). The latter two are subseries of RFCs (more about these in the next section).

The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols. It is the clearinghouse to assign and coordinate use of numerous Internet protocol parameters. The Internet protocol suite contains numerous parameters, such as Internet addresses, domain names, autonomous system numbers (used in some routing protocols), protocol numbers, port numbers, MIB object identifiers (including private enterprise numbers), and many others. The common use of Internet protocols by the Internet community requires that these values be assigned uniquely. It is the task of IANA to make those unique assignments as requested and to maintain a registry of currently assigned values.

4.3.2 Internet Documents

Originally, RFC was just what the name implies—Request for Comments. Early RFCs were messages between ARPANET architects about how to resolve certain problems. Over the years, RFC has become more formal. It had reached the point that they were being cited as standards, even when they were not. To help clear up some confusion, there are now two special subseries within the RFCs: FYIs and STDs. The "For Your Information" RFC subseries was created to document overviews and topics that are introductory. Frequently, FYIs are created by groups within the IETF User Services Area. The STD RFC subseries was created to identify those RFCs that do in fact specify Internet standards. Every RFC, including FYIs and STDs, has an RFC number by which they are indexed and can be retrieved. FYIs and STDs have FYI numbers and STD numbers, respectively, in addition to RFC numbers. This makes it easier for a new Internet user, for example, to find all of the helpful, informational documents by looking for the FYIs among all the RFCs. If an FYI or STD is revised, its RFC number will change, but its FYI or STD number will remain constant for ease of reference.

RFC documents are available in public libraries and can be accessed via the Internet. Some sources that are in the public domain to access RFC and other Internet documents are:

ftp://ftp.internic.net/rfc
ftp://nic.mil/rfc
ftp.nic.it
http://nic.internic.net/

A novice to SNMP management could easily be confused as to which RFC document refers to what, namely, SMI, MIB, and SNMP, etc. It is confusing because the management field and associated documents are continuously evolving. Figure 4.4 portrays a high-level view of various document paths and

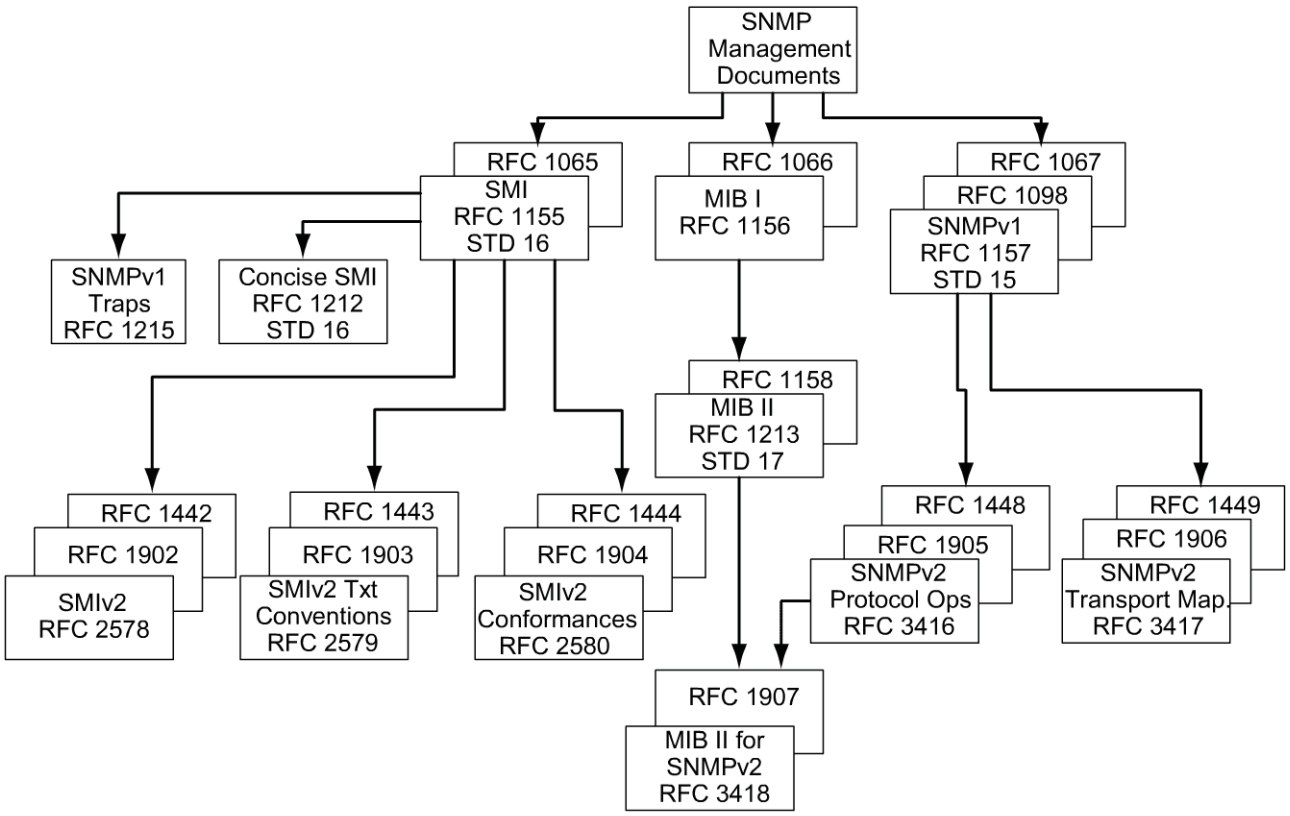


Figure 4.4 SNMP Document Evolution

documents that are relevant to SNMPv1 and SNMPv2. Documents associated with SNMPv3 will be described in Chapter 7. It is not intended to be a complete list, but to identify major core documents. There are three series of RFC and STD documents. They are: SMI, MIB, and SNMP Protocol. There are three standard documents, STD 15, 16, and 17 that have been approved by the IETF. STD 15/RFC 1157 defines the SNMP protocol. RFCs 1905, on protocol operations, and 1906, on transport mappings, are expanded updates of RFC 1157. These have been updated to RFC 1448 and RFC 1449 and subsequently, with the evolution of SNMPv3, to RFC 3416 and RFC 3417, respectively. In Figure 4.4, RFCs in the back of the cascades are earlier versions of the draft that have become obsolete. For example, RFC 1448 has been replaced by RFC 1905.

Structure of Management Information (SMI) forms the contents of RFC 1155, shown in Figure 4.4. A more concise version of SMI is given in RFC 1212 and is a supplement to RFC 1155. They both comprise STD 16 document. RFC 1155 did not address trap events, which is covered in RFC 1215.

SMIv2 is next in the evolution of SMI specifications, which are covered as STD 58 with the three documents RFC 2578–RFC 2580 describing SMIv2 data definition language, textual conventions, and conformance, respectively.

MIB has gone through a few iterations. RFC 1213/STD 17 is the version that is currently in use. It is backward compatible with MIB I specified in RFC 1156, which is obsolete now. Legacy systems that have implemented MIB I can continue to be used with MIB II implementation.

SNMP protocol has gone through modification and is part of SNMPv2. RFC 1907 is an early version of MIB II for SNMPv2 and the latest version is RFC 3418, which has gone through only minor changes from RFC 1907.

4.4 SNMP MODEL

We described an example of a managed network in Section 4.1. We saw that numerous management functions were accomplished in that example. We will now address how this is done in SNMP management. An NMS acquires a new network element through a management agent or monitors the ones it has acquired. There is a relationship between manager and agent. Since one manager is responsible for managing the designated functions of many agents, it is hierarchical in structure. The infrastructure of the manager–agent and the SNMP architecture that it is based on form the organization model.

Information is transmitted and is received by both the manager and the agent. For example, when a new network element with a built-in management agent is added to the network, the discovery process in the network manager broadcasts queries and receives positive response from the added element. The information must be interpreted both semantically and syntactically by the agent and the manager. We covered the syntax, ASN.1, in Section 3.7. Definition of semantics and syntax form the basis of the information model. We present a detailed definition of a managed object, rules for the SMI, and a virtual information database, MIB, which groups managed objects and provides a relational framework.

Communication between the manager and agents has to happen before information can be exchanged. The TCP/IP protocol suite is used for the transport mechanism. SNMP is defined for the application layer protocol and will be presented in Chapter 5.

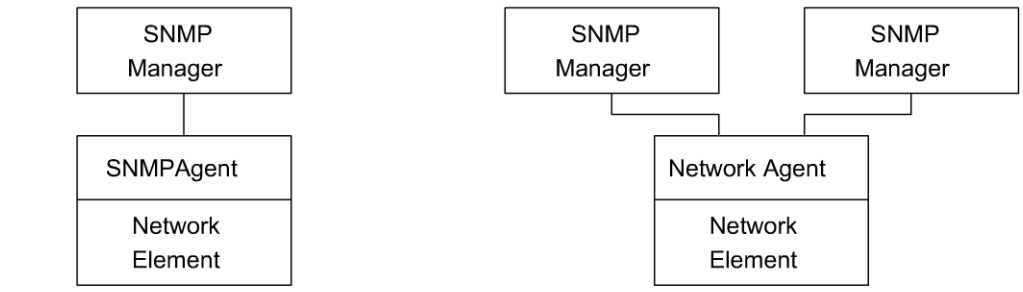
Functions and services are not explicitly addressed in SNMP management. Security management is covered in the administration model as part of communication. Services are covered as part of SNMP operations.

The organization model, which has gone through an evolutionary process, is described in the next section.

4.5 ORGANIZATION MODEL

The initial organization model of SNMP management is a simple two-tier model. It consists of a network agent process, which resides in the managed object, and a network manager process, which resides in the NMS and manages the managed object. This is shown in Figure 4.5(a). Both the manager and the agent are software modules. The agent responds to any management system that communicates with it using SNMP. Thus, multiple managers can interact with one agent as shown in Figure 4.5(b)

We can question the need of multiple managers in a system when it is easy to monitor all objects in a network with standard messages. However, to configure a system in detail, more intimate knowledge of the object is needed, and hence an NMS provided by the same vendor would have more capabilities than another vendor's NMS. Thus, it is common practice to use an NMS to monitor a network of multiple



(a) One Manager–One Agent Model (b) Multiple Managers–One Agent Model

Figure 4.5 Two-Tier Organization Model

vendor products, and several vendors' NMSs to configure respective network elements. Further, during fault tracking, a vendor's NMS can probe in more depth the source of failure—even to the level of identification of a component on a printed circuit board.

In two-tier models, the network manager receives raw data from agents and processes them. It is sometimes beneficial for the network manager to obtain pre-processed data. For example, we may want to look at traffic statistics, such as input and output packets per second, at an interface on a node as a function of time. Alternatively, we may want to get the temporal data of data traffic in a LAN. Instead of the network manager continuously monitoring events and calculating the information—for example, data rate—an intermediate agent called Remote Monitoring (RMON) is inserted between the managed object and the network manager. This introduces a three-tier architecture as shown in Figure 4.6. The network manager receives data from managed objects, as well as from the RMON agent about the managed objects. The RMON function, implemented in a distributed fashion on the network, has greatly increased the centralized management of networks.

A pure SNMP management system consists of SNMP agents and SNMP managers. However, an SNMP manager can manage a network element, which does not have an SNMP agent. Figure 4.7 shows

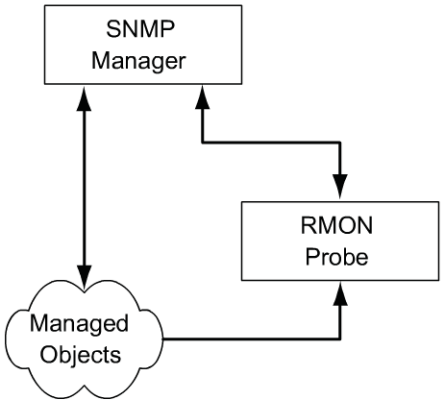


Figure 4.6 Three-Tier Organization Model

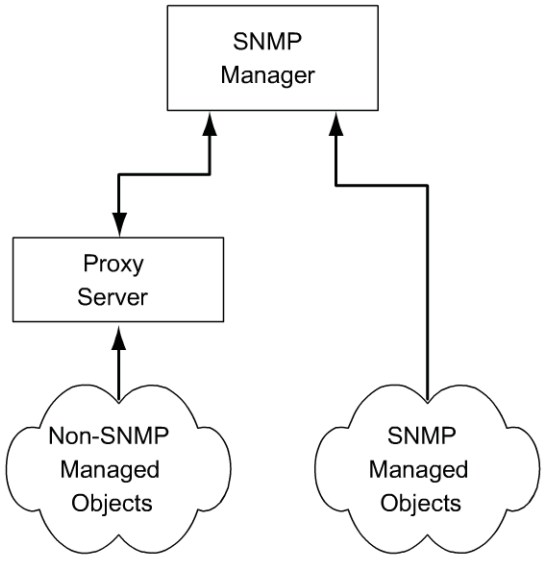


Figure 4.7 Proxy Server Organization Model

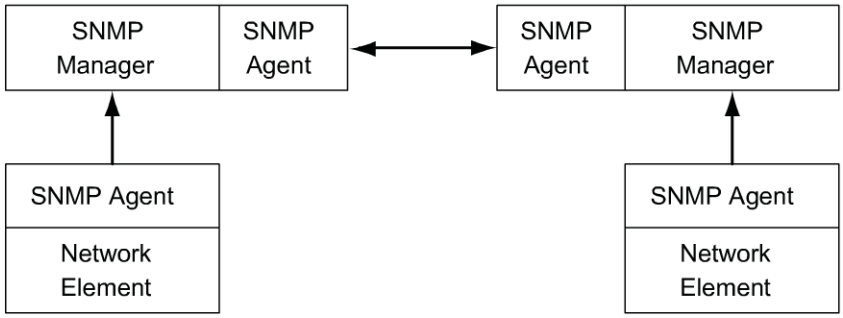


Figure 4.8 NMS Behaving as a Manager and an Agent

the organizational model for this case. This application occurs in many situations, such as legacy systems management, telecommunications management network, managing wireless networks, etc. In all these cases, they are part of an overall network that have to be managed on an integrated basis. As an example in a legacy case, we may want to manage outside plant and customer premises equipment for a Hybrid Fiber Coax (HFC) access system in broadband services to home. There are amplifiers on the outside cable plant, which do not have SNMP agents built in them. The outside cable plant uses some existing cable technology and has monitoring tools built into it, as for example transponders that measure various amplifier parameters. Information from the amplifiers could be transmitted to a central (head end) location using telemetry facilities. We can have a proxy server at the central location that converts data into a set that is SNMP compatible and communicates with the SNMP manager.

An SNMP management system can behave as an agent as well as a manager. This is similar to client–server architecture, where a host can function as both a server and a client (see Figure 1.8). In Figure 4.6, RMON, while collecting data from network objects, performs some functions (network monitoring) of a network manager. However, pre-processed data by RMON may be requested by the network manager or sent unsolicited by RMON to the network manager to integrate with the rest of the network data and to display it to the user. In the latter situation, RMON acts as a network agent. Another example of a system acting as both an agent and a manager is when two NMSs managing two autonomous networks exchange information with each other when the networks are connected via a gateway. This model is presented in Figure 4.8 and is applicable to two telecommunication service providers managing their respective wide area networks. To provide end-to-end service to customers, service providers may need to exchange management information between them.

4.6 SYSTEM OVERVIEW

Now that we have learned the relationship between the network (management) agent and manager and the different ways they can be configured, let us consider SNMP management from a system point of view. We have opted to do this prior to discussing details of the other three models—information, communication, and functional, because it would help us to understand them better if we have the big picture first.

Figure 4.9 shows SNMP network management architecture. It portrays the data path between the manager application process and the agent application process via the four transport function protocols—UDP, IP, Data Link Control (DLC), and Physical (PHY). The three application layers above the transport layer are integrated in the SNMP process.

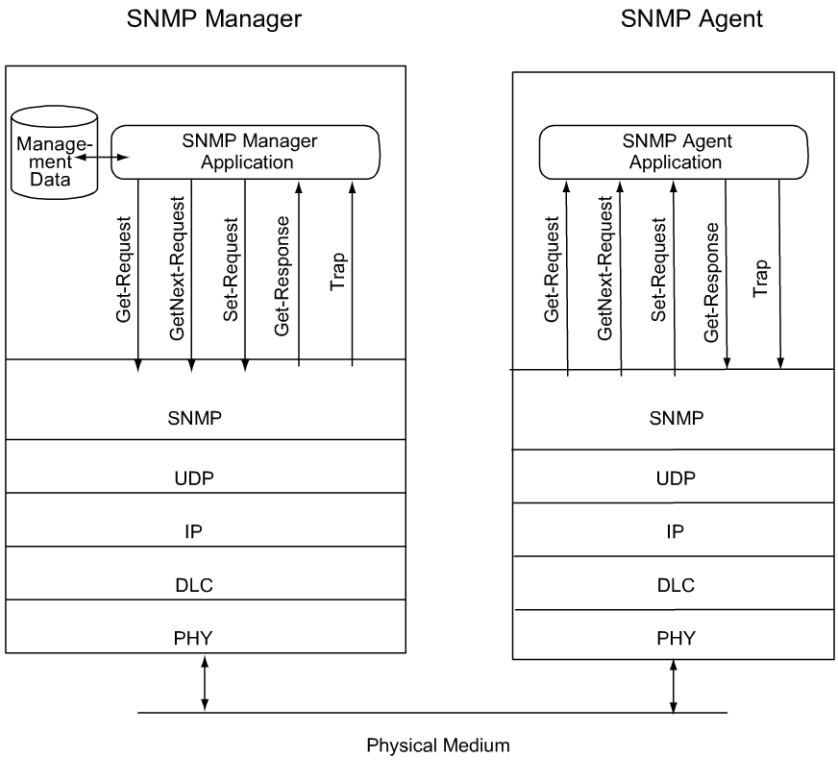


Figure 4.9 SNMP Network Management Architecture

As we stated in Chapter 1, the Internet is only concerned with the TCP/IP suite of protocols and does not address layers above or below it. Thus, layers 1 (PHY) and 2 (DLC) in the transport layers can be anything of users' choice. In practice, SNMP interfaces to the TCP/IP with UDP as the transport layer protocol.

RFC 1157 describes SNMP system architecture. It defines SNMP "by which management information for a network element may be inspected or altered by logically remote users." Two companion RFCs are RFC 1155, which describes the structure and identification of management information, and RFC 1156, which addresses the information base that is required for management.

As the name implies, SNMP protocol has been intentionally designed to be simple and versatile; this surely has been accomplished as indicated by its success. Communication of management information among management entities is realized through exchange of just five protocol messages. Three of these (get-request, get-next-request, and set-request) are initiated by the manager application process. The other two messages, get-response and trap, are generated by the agent process. Message generation is called an event. In the SNMP management scheme, the manager monitors the network by polling the agents as to their status and characteristics. However, efficiency is increased by agents generating unsolicited alarm messages, i.e., traps. We will summarize the messages here and describe structures associated with their Packet Data Units (PDUs) later. RFC 1157 defines the original specifications.

The get-request message is generated by the management process requesting the value of an object. The value of an object is a scalar variable. System group parameters in Figure 4.2 are single-instance values and are obtained using the get-request message.

The get-next-request, or simply called get-next, is very similar to get-request. In many situations, an object may have multiple values because of multiple instances of the object. For example, we saw in

Figure 4.3 that an interface could have multiple addresses associated with a given row. Another example is the routing table of a router, which has multiple values (instances) for each object. In such situations, get-next-request obtains the value of the next instance of the object.

The set-request is generated by the management process to initialize or reset the value of an object variable. The configuration parameters in Figure 4.2 that are settable can be set using the set-request message.

The get-response message is generated by an agent process. It is generated only on receipt of a get-request, get-next-request, or set-request message from a management process. The get-response process involves filling the value of the requested object with any success or error message associated with the response.

The other message that the agent generates is trap. A trap is an unsolicited message generated by an agent process without any message or event arriving from the manager process. It occurs when it observes the occurrence of a preset parameter in the agent module. For example, a node can send traps when an interface link goes up or down. Or, if a network object has a threshold value set for a parameter, such as the maximum number of packets queued up, a trap could be generated and transmitted by the agent application whenever the threshold is crossed in either direction.

The SNMP manager, which resides in the NMS, has a database that polls managed objects for management data. It contains two sets of data—one on information about the objects, MIB, and a second on the values of the objects. These two are often confused with each other. MIB is a virtual data (information) base and is static. In fact, it needs to be there when an NMS discovers a new object in the network. It is compiled in the manager during implementation. If information about the managed object is not in the manager, it could still detect the object but would mark it as unidentifiable. This is because the discovery process involves a broadcast PING command by the NMS and responses to it from network components. Thus, a newly added network component would respond if it has a TCP/IP stack that normally has a built-in ICMP. However, the response contains only the IP address. MIB needs to be implemented in both the manager and the agent to acquire the rest of the information, such as the system group information shown in Figure 4.2.

The second database is dynamic and contains measured values associated with the object. This is a true database. While MIB has a formalized structure, the database containing actual values can be implemented using any database architecture chosen by the implementers.

It is worth noting in Figure 4.9 that the SNMP manager has a database, which is the physical database, and the SNMP agent does not have a physical database. However, both have MIBs, which are compiled into the software module and are not shown in the figure.

4.7 INFORMATION MODEL

The information model deals with SMI and MIB, which are discussed in the following subsections.

4.7.1 Introduction

Figure 4.9 shows the information exchange between the agent and the manager. In a managed network, there are many managers and agents. For information to be exchanged intelligently between manager and agent processes, there has to be common understanding on both the syntax and semantics. The syntax used to describe management information is ASN.1 and a general introduction to it was given in Chapter 3. In this section, we will address SNMP-specific syntax and semantics of management information.

We discussed the types of messages in the previous section and will discuss more in Chapter 5 when we consider the communication model. In this section we will address the specification and organizational aspects of managed objects. This is called the Structure of Management Information, SMI, and is defined in RFC 1155. Specifications of managed objects and the grouping of, and relationship between, managed objects are addressed in the MIB [RFC 1213].

There are generic objects that are defined by IETF and can be managed by any SNMP-compatible NMS. Objects that are defined by private vendors, if they conform to SMI defined by RFC 1155 and MIB specified by RFC 1213, can also be managed by SNMP-compatible NMSs. There are other RFCs that address specialized network objects, such as FDDI [RFC 1285], OSPF [RFC 1253], ATM [RFC 1695], etc. Private vendor objects are specified in private MIBs provided by vendors for their specific products.

4.7.2 Structure of Management Information

A managed object can be considered to be composed of an object type and an object instance, as shown in Figure 4.10. SMI is concerned only with the object type and not the object instance. That is, the object instance is not defined by SMI. For example, Figures 4.2(a) and (b) present data on two 3Com hubs. They are both identical hubs, except for a minor software release difference. The object types associated with both hubs are represented by the identical object ID, iso.org.dod.internet.private.enterprises.43.1.8.5. Hub 1 with an IP address 172.16.46.2 is an instance of the object.

Figure 4.11 shows the situation where there are multiple instances of an object type. In Figures 4.2(a) and (b), hub 1 with an IP address 172.16.46.2 and hub 2 with an IP address 172.16.46.3 are two instances of the object.

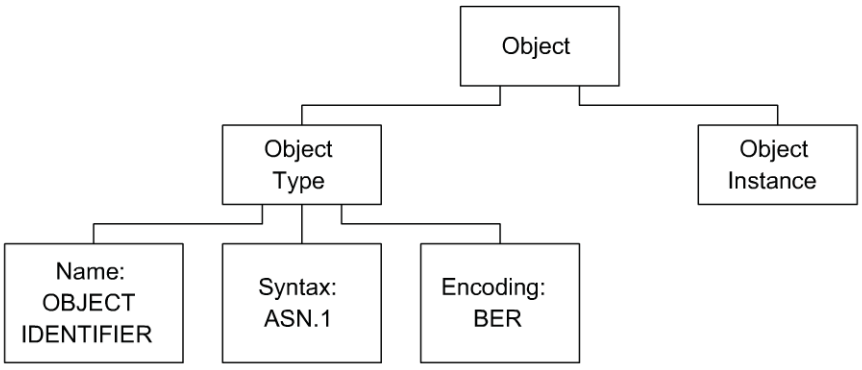


Figure 4.10 Managed Object: Type and Instance

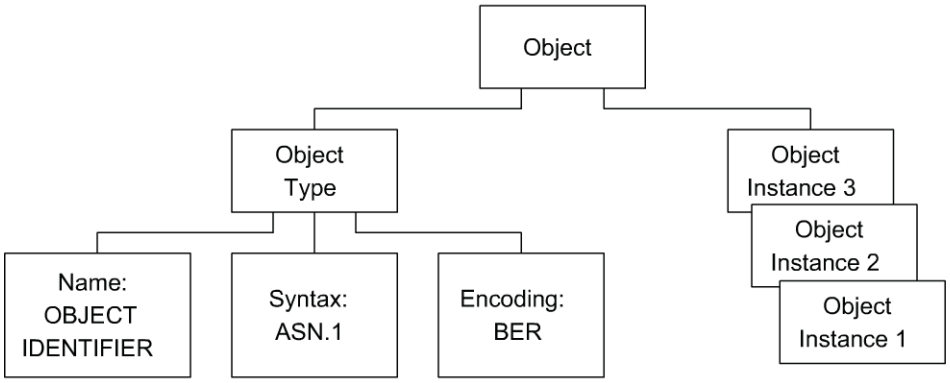


Figure 4.11 Managed Object: Type with Multiple Instances

A managed object need not be just a network element, it could be any object. For example, the Internet as an organization has an object name, "internet," with OBJECT IDENTIFIER 1.3.6.1. Of course, there can only be one instance of it! Thus, a managed object is only a means of identifying an object, whether it is physical or abstract.

The object type, which is a data type, has a name, a syntax, and an encoding scheme as discussed in Section 3.7. The name is represented uniquely by a descriptor and an object identifier. The syntax of an object type is defined using the abstract data structure ASN.1. Basic encoding rules (BER) have been adopted as the encoding scheme for transfer of data types between agent and manager processes, as well as between manager processes. We will next discuss each of these for SNMP-managed objects in detail.

Names. Every object type, i.e., every name, is uniquely identified by a DESCRIPTOR and an associated OBJECT IDENTIFIER. DESCRIPTOR and OBJECT IDENTIFIER are in uppercase since they are ASN.1 keywords. The DESCRIPTOR defining the name is mnemonic and is all in lowercase letters—at least it begins with lowercase letters, as we just described the Internet object as "internet." Since it is mnemonic and should be easily readable, uppercase letters can be used as long as they are not the beginning letter. For example, the object IP address table is defined as *ipAddrTable*. OBJECT IDENTIFIER is a unique name and number in the Management Information Tree (MIT), as we discussed in Section 3.4.1. We will henceforth use the term Management Information Base (MIB) for the Internet MIT. Thus, the Internet MIB has its OBJECT IDENTIFIER 1.3.6.1, as shown in Figure 3.5. It can also be defined in a hybrid mode, for example,

```
internet OBJECT IDENTIFIER ::= {iso org(3) dod(6) 1 }.
```

Information inside the curly brackets can be represented in various ways. This is shown in Figure 4.12. We can use any combination of the unique name and the unique node number on the management tree.

Any object in the Internet MIB will start with the prefix 1.3.6.1 or *internet*. For example, there are four objects under the *internet* object. These four objects are defined as:

```
directory       OBJECT IDENTIFIER ::= {internet 1}
mgmt            OBJECT IDENTIFIER ::= {internet 2}
experimental    OBJECT IDENTIFIER ::= {internet 3}
private         OBJECT IDENTIFIER ::= {internet 4}
```

The first line in this example states that the object, directory, is defined as the first node under the object internet. The four subnodes under the "internet" node are shown in Figure 4.13. We will discuss objects in the MIB tree in the next section.

The *directory*(1) node is reserved for future use of OSI Directory in the Internet. The *mgmt*(2) node is used to identify all IETF recommended and IAB-approved subnodes and objects. As of now the only

```
internet OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) internet(1) }
internet OBJECT IDENTIFIER ::= { 1 3 6 1 }
internet OBJECT IDENTIFIER ::= { iso org  dod internet }
internet OBJECT IDENTIFIER ::= { iso org  dod(6) internet(1) }
internet OBJECT IDENTIFIER ::= { iso(1) org(3) 6 1 }
```

Figure 4.12 Different Formats of Declaration of OBJECT IDENTIFIER

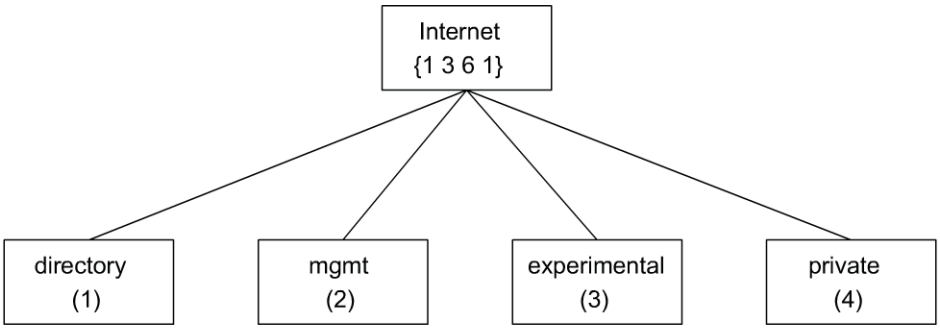


Figure 4.13 Subnodes under Internet Node in SNMPv1

node connected directly to {internet 2} is *mib-2*. As we said earlier, MIB-2 is a superset of MIB-1, and hence *mib-2* is the only node under {mgmt} as shown below:

```
mib-2          OBJECT IDENTIFIER ::= {mgmt 1}
```

The *experimental*(3) node was created to define objects under IETF experiments. For example, if IANA has approved a number 5 for an experimenter, we would use the OBJECT IDENTIFIER {experimental 5}.

The last node is *private*(4). This is a heavily used node. Commercial vendors can acquire a number under enterprises(1), which is under the private(4) node. Thus, we have

```
enterprises    OBJECT IDENTIFIER ::= {private 1}
```

or

```
enterprises    OBJECT IDENTIFIER ::= {1 3 6 1 4 1}
```

Figure 4.14 shows an example of four commercial vendors—Cisco, HP, 3Com, and Cabletron who are registered as nodes 9, 11, 43, and 52, respectively, under enterprises(1). Nodes under any of these nodes are entirely left to the discretion of the vendors.

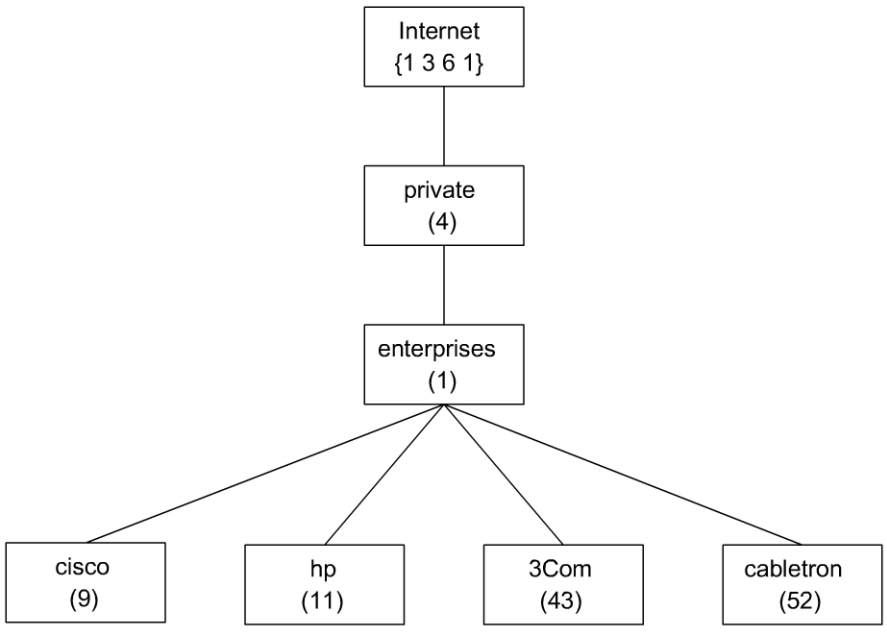


Figure 4.14 Private Subtree for Commercial Vendors

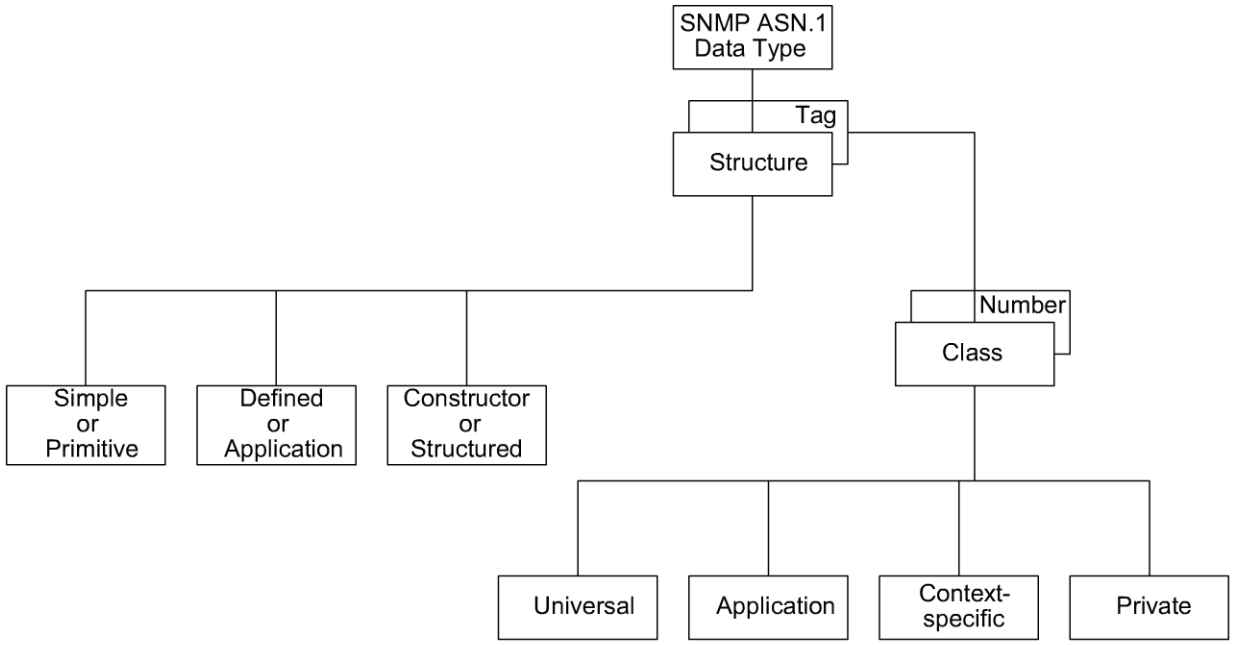


Figure 4.15 SNMP ASN.1 Data Type

Syntax. ASN.1 syntax that was introduced in Section 3.7 is used to define the structure of object types. Not all constructs of ASN.1 are used in TCP/IP-based SNMP management. Figure 4.15 shows the TCP/IP-based ASN.1 data type. It is very similar to Figure 3.15, but only has three categories under structure.

The three structural types shown in Figure 4.15 are simple, constructor, and defined types, as defined in RFC 1155. Other common terms used for these are primitive (or atomic), structured, and application types, respectively, as shown in Figure 3.9. The tagged type is not explicitly used in TCP/IP management, although the IMPLICIT and EXTERNAL keywords are utilized for derived application data types.

SNMP ASN.1 data types are listed in Table 4.1. All data types except SEQUENCE and SEQUENCE OF are called base types.

Primitive or simple types are atomic in nature and are: INTEGER, OCTET STRING, OBJECT IDENTIFIER, and NULL. These are also referred to as non-aggregate types.

INTEGER has numerous variations based on the sign, length, range, and enumeration. The reader is referred to Perkins and McGinnis [1997] for a detailed presentation on the subject. When the integer value is restricted by a range, it is called a subtype, as presented in the comments column of Table 4.1, as INTEGER (n1..nN).

The data type ENUMERATED was specified in Section 3.6.2 as a special case of INTEGER data type. In SNMP management, it is specified as INTEGER data type with labeled INTEGER values. The following example of error-status in GetResponse associated with GetRequest-PDU illustrates the use of it. Each enumerated INTEGER has a name associated with it:

```
error-status INTEGER {
        noError(0)
        tooBig(1)
        genErr(5)
        authorizationError(16)
}
```

Table 4.1 SNMP-based ASN.1 Data Type Structure

STRUCTURE	DATA TYPE	COMMENTS
Primitive types	INTEGER	Subtype INTEGER (n1..nN)
		Special case: Enumerated INTEGER type
	OCTET STRING	8-bit bytes binary and textual data
		Subtypes can be specified by either range or fixed
	OBJECT IDENTIFIER	Object position in MIB
	NULL	Placeholder
Defined types	NetworkAddress	Not used
	IpAddress	Dotted decimal IP address
	Counter	Wrap-around, non-negative integer, monotonically increasing, max 2^32 -1
	Gauge	Capped, non-negative integer, increase or decrease
	TimeTicks	Non-negative integer in hundredths of second units
	Opaque	Application-wide arbitrary ASN.1 syntax, double-wrapped OCTET STRING
Constructor types	SEQUENCE	List maker
	SEQUENCE OF	Table maker

Any non-zero value indicates the type of error encountered by the agent in responding to a manager's message. As a convention, the value 0 is not permitted in the response message. Thus, a noError message is filled with NULL.

The OCTET STRING data type is used to specify either binary or textual information that is 8 bits long. Just as in INTEGER data type, a subtype in OCTET STRING can be specified. In fact, the subtype value can either be ranged, fixed, or a choice between them. Some examples of the subtype are:

```
OCTET STRING (SIZE 0..255)
OCTET STRING (SIZE 8)
OCTET STRING (SIZE 4 | 8)
OCTET STRING (SIZE 0..255 | 8)
```

The combination keyword OBJECT IDENTIFIER, as we discussed before, is the object position in the MIB. The fourth primitive type listed in Table 4.1 is NULL and is also a keyword. SNMPv1 keywords are listed in Table 4.2.

The second category of data types shown in Figure 4.15 and Table 4.1 consists of **defined types**. These are application-specific data types, and are also SNMP-based types. They are defined using primitive types. The primitive types used are NetworkAddress (not used in SNMP management), IpAddress, Counter, Gauge, and TimeTicks. The base type, Opaque, is used to specify octets of binary information. It is intended for adding new base types to extend SNMP SMI. Other application-wide data types can be constructed as long as they are IMPLICITly defined using these application data types.

Table 4.2 SNMPv1 Keywords

ACCESS
BEGIN
CHOICE
Counter
DEFINITIONS
DEFVAL
DESCRIPTION
END
ENTERPRISE
FROM
Gauge
IDENTIFIER
IMPORTS
INDEX
INTEGER
IpAddress
NetworkAddress
OBJECT
OBJECT-TYPE
OCTET
OF
Opaque
REFERENCE
SEQUENCE
SIZE
STATUS
STRING
SYNTAX
TRAP-TYPE
TimeTicks
VARIABLES

NetworkAddress is a choice of the address of the protocol family. For us, it is the TCP/IP-base Internet family, which uses the base type IpAddress.

IpAddress is the conventional four groups of dotted decimal notation of IPv4; for example, 190.146.252.255. The 32-bit string is designated as OCTET STRING of length 4 in network byte order.

Counter is an application-wide data type and is a non-negative integer. It can only increase in value up to a maximum of $2^{32}-1$ (4,294,967,295) and then wraps around starting from 0. The counter type is useful for defining values of data types that continually increase, such as input packets received on an interface or output packet errors on an interface.

The data type **Gauge** is also a non-negative integer, but its value can move either up or down. It pegs at its maximum value of $2^{32}-1$ (4,294,967,295). Gauge is used for data types whose value increases or decreases, such as the number of interfaces that are active in a router or hub.

TimeTicks is a non-negative integer and measures time in units of hundredths of a second. Its value indicates in hundredths of a second the number of units of time between the current instant and the time it was initialized to 0. The maximum value is $2^{32}-1$ (4,294,967,295). The system up time in Figure 4.2 is an example of this.

Opaque is an application-wide data type that supports the capability to pass arbitrary ASN.1 syntax. It is used to create more data types based on previously defined data types. This is extensively used in private vendors defining new data types in their products. When it is encoded, it is double wrapped, meaning the TLV (tag, length, and value) for the new definition is wrapped around the TLV of the previously defined type. Its size is undefined in SNMPv1, which causes some problem in its implementation. It is limited in SNMPv2.

The Opaque data type can be defined both IMPLICITly and EXPLICITly. By use of EXTERNAL type, encoding other than ASN.1 may be used in opaquely encoded data.

The third and last type of structure shown in Figure 4.15 is **constructor** or **structured type**. SEQUENCE and SEQUENCE OF are the only two constructor data types in Table 4.1 that are not base types. They are used to build lists and tables. Note that the constructs SET and SET OF, which are in ASN.1, are not included in the SNMP-based management syntax. SEQUENCE is used to build a list and SEQUENCE OF is used to build a table. We can conceptualize the list as values in a row of a table.

The syntax for list is

SEQUENCE { <type1>, <type2>,…, <typeN> }

where each type is one of ASN.1 primitive types.

The syntax for table is

SEQUENCE OF <entry>

where <entry> is a list constructor.

Illustrations of building list and table are shown in Figures 4.16(a) and (b). Figure 4.16(a) shows the object *ipAddrEntry* as an entry that is created from a list of objects. The list of objects in Figure 4.16(a) is 1 through 5 in the table. They are all basic types and each row of an object has the object name, OBJECT IDENTIFIER and *ObjectSyntax*. For example, object 1 on row 1 is the IP address defined as *ipAdEntAddr*. It has an OBJECT IDENTIFIER {ipAddrEntry 1} and syntax IpAddress. Note that there are two data types (ObjectSyntax) in the table, namely IpAddress and INTEGER. Thus, data types can be mixed in building a list. However, they are all basic data types and not constructor types.

The sixth object in the table is the object *ipAddrEntry* and is made up of the list of the first five objects. Construction for that is a SEQUENCE data type structure as shown. In Figure 4.16(a), the object *ipAdEntReasmMaxSize* has the syntax INTEGER(0..65535), which denotes that it is a subtype and the integer can take on values in the range from 0 to 65535.

Figure 4.16(b) shows the seventh object, *ipAddrTable*. It is node 20 under ip node and has a SEQUENCE OF construct. The *ipAddrTable* table is made up of instances of *ipAddrEntry* object.

Encoding. SNMPv1 has adopted BER with its TLV for encoding information to be transmitted between agent and manager processes. We covered this in Section 3.8 and illustrated a few ASN.1 data types. SNMP data types and tags are listed in Table 4.3. Encoding rules for various types follow.

OBJECT IDENTIFIER is encoded with each subidentifier value encoded as an octet and concatenated in the same order as in the object identifier. Since a subidentifier could be longer than an octet length, the

	Object	OBJECT IDENTIFIER	ObjectSyntax
1	ipAdEntAddr	{ipAddrEntry 1}	IpAddress
2	ipAdEntIfIndex	{ipAddrEntry 2}	INTEGER
3	ipAdEntNetMask	{ipAddrEntry 3}	IpAddress
4	ipAdEntBcastAddr	{ipAddrEntry 4}	INTEGER
5	ipAdEntReasmMaxSize	{ipAddrEntry 5}	INTEGER
6	ipAddrEntry	{ipAddrTable 1}	SEQUENCE

```
List:    IpAddrEntry    ::=
                SEQUENCE    {
                        ipAdEntAddr                IpAddress
                        ipAdEntIfIndex             INTEGER
                        ipAdEntNetMask             IpAddress
                        ipAdEntBcastAddr           INTEGER
                        ipAdEntReasmMaxSize        INTEGER (0..65535)
                                }
```

(a) Managed Object IpAddrEntry as a List

	Object Name	OBJECT IDENTIFIER	Syntax
7	ipAddrTable	{ip 20}	SEQUENCE OF

```
Table:  IpAddrTable ::=
                SEQUENCE OF IpAddrEntry
```

(b) Managed Object ipAddrTable as a Table

Figure 4.16 Example of Building a List and a Table for a Managed Object

Table 4.3 SNMP Data Types and Tags

TYPE	TAG
OBJECT IDENTIFIER	UNIVERSAL 6
SEQUENCE	UNIVERSAL 16
IpAddress	APPLICATION 0
Counter	APPLICATION 1
Gauge	APPLICATION 2
TimeTicks	APPLICATION 3
Opaque	APPLICATION 4

most significant bit (8^{th} bit) is set to 0, if the subidentifier is only one octet long. The 8^{th} bit is set to 1 for the value that requires more than one octet and indicates more octet(s) to follow. An exception to the rule of one or more octets for each subidentifier is the specification of the first two subidentifiers. For

example, *iso(1)* and *standard(3)* {1 3}, are coded as 43 in the first octet of the value. As an illustration, let us consider the object identifier *internet* {1 3 6 1}. The first octet of the TLV is the UNIVERSAL 6 tag, and the second octet defines the length of the value, which consists of three octets (43, 6, and 1). Thus, the encoded format is:

00000110 00000011 00101011 00000110 00000001

IP Address is encoded as straight octet strings. Counter, Gauge, and TimeTicks are coded as integers. Opaque is OCTET STRING type.

4.7.3 Managed Objects

In Chapter 3 we briefly looked at the perspective of a managed object in an SNMP management model and compared it to the OSI model. We will now specify in detail the SNMP data type format that would serve the basis for defining managed objects. We will address managed objects in the MIB in Section 4.7.4.

Structure of Managed Objects. Managed object, as we saw in Section 3.4.2, has five parameters. They are textual name, syntax, definition, access, and status as defined in RFC 1155. For example, *sysDescr* is a data type in the MIB that describes a system. Specifications for the object that describes a system are given in Figure 4.17.

As we notice in Figure 4.17, the **textual name** for an object type is mnemonic and is defined as OBJECT DESCRIPTOR. It is unique and is made up of a printable string beginning with a lowercase letter, *sysDescr*, in our example. OBJECT DESCRIPTOR defines only the object type, which is a data type. We will henceforth use the term *object type* and not *data type* when referring to a managed object. OBJECT DESCRIPTOR does not specify instances of a managed object. Thus, it describes what type of object it is and not the occurrence or instantiation of it, as we pointed out in Section 4.7.2. In Figures 4.2(a) and (b), the system description for the two hubs is 3Com LinkBuilder FMS with an appropriate software version. They both could be of the same software version and hence could be identical. Identification of each instance is left to the specific protocol that is used, and is not part of the specifications of either SMI or MIB. Thus, instances of the two hubs in Figures 4.2(a) and (b) are identified with their respective IP addresses, 172.16.46.2 and 172.16.46.3.

Associated with each OBJECT DESCRIPTOR is an OBJECT IDENTIFIER, which is the unique position it occupies in the MIB. In Figure 4.17, *sysDescr* is defined by OBJECT IDENTIFIER {system 1}.

OBJECT:	
sysDescr:	{ system 1 }
Syntax:	DisplayString (SIZE (0..255))
Definition:	"A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating system, and networking software. It is mandatory that this contain only printable ASCII characters."
Access:	read-only
Status:	mandatory

Figure 4.17 Specifications for System Description

Syntax is the ASN.1 definition of the object type. The syntax of s*ysDescr* is OCTET STRING.

A **definition** is an accepted textual description of the object type. It is a basis for the common language or semantics to be used by all vendors. It is intended to avoid confusion in the exchange of information between the managed object and the management system, as well as between various NMSs.

Access is the specification for the privilege associated with accessing the information. It is one of read-only, read-write, write-only, or not-accessible. The first two choices are obvious and the third choice, not-accessible, is applicable, for example, in specifying a table. We access the values of the entries in the table and not the table itself, and hence it is declared not-accessible. The access for *sysDescr* is read-only. Its value is defined by the system vendor during the manufacturing process.

Status specifies whether the managed object is current or obsolete. A managed object, once defined, can only be made obsolete and not removed or deleted. If it is current, the implementation of it is specified as either mandatory or optional. Thus, the three choices for status are mandatory, optional, and obsolete. The status for *sysDescr* is mandatory.

Related objects can be grouped to form an **aggregate object type**. In this case the objects that make up the aggregate object type are called subordinate object types. The **subordinate object type** could either be simple (primitive type) or an aggregate type. However, it should eventually be made up of simple object types.

Macros for Managed Objects. In order to encode the above information on a managed object to be processed by machines, it has to be defined in a formalized manner. This is done using macros. Figure 4.18(a) shows a macro where an object type is represented in a formal way [RFC 1155]. A macro always starts with the name of the type—in this case, OBJECT-TYPE—followed by the keyword MACRO, and then the definition symbol. The right side of the macro definition always starts with BEGIN and ends with END.

The body of the macro module consists of three parts: type notation, value notation, and supporting productions. TYPE NOTATION defines the data types in the module and VALUE NOTATION defines the name of the object. Thus, in the example of Figure 4.18, the notations SYNTAX, ACCESS, and STATUS define the data types Object Syntax, Access, and Status. The notation for value specifies the Object Name. Supporting productions in Figure 4.18 define the allowed values for access and status. Access can only be one of any of the four options: read-only, read-write, write-only, or not-accessible. Allowed values for Status are mandatory, optional, or obsolete.

```
OBJECT-TYPE MACRO ::=
BEGIN
        TYPE NOTATION ::= "SYNTAX" TYPE (TYPE ObjectSyntax)
                "ACCESS" Access
                "STATUS" Status
        VALUE NOTATION ::= value (VALUE ObjectName)
        Access ::=  "read-only" | "read-write" | "write-only" | "not-accessible"
        Status ::= "mandatory" | "optional" | "obsolete"
END
```

(a) An OBJECT-TYPE Macro [RFC 1155]

Figure 4.18 Scalar OBJECT-TYPE Macro and Example

```
sysDescr OBJECT-TYPE
        SYNTAX Display String (SIZE (0..255))
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION
                "A textual description of the entity. This value should include the full
                name and version identification of the system's hardware type,
                software operating system, and networking software. It is
                mandatory that this contain only printable ASCII characters."
::= {system 1 }
```

(b) A Scalar or Single Instance Macro: sysDescr [RFC 1213]

Figure 4.18 (continued)

Figure 4.18(b) [RFC 1213] shows the application of the macro to a scalar, single-instance managed object, *sysDescr*, which is one of the components of the system group in the MIB, as we shall see in the next section. Its OBJECT IDENTIFICATION is {system 1}. DESCRIPTION defines the textual description of the object.

Aggregate Object. An aggregate object is a group of related objects. Figure 4.19 shows an example of an aggregate managed object, *ipAddrTable*, which we briefly considered as an example of structured data type in Figure 4.16. This is the IP address table that defines the IP address for each interface of the managed object. Objects 1 through 5 represent simple data types that make up an entry in a table. The textual name of the entry is *ipAddrEntry*. Thus, object 1 with the OBJECT DESCRIPTOR, *ipAdEntAddr*, is the first element of the entry, *ipAddrEntry*, and is given the unique OBJECT IDENTIFICATION, {ipAddrEntry 1}. This represents the IP address and has the syntax IpAddress, a keyword listed in Table 4.2. The access privilege to it is read-only and every managed object and management system is required to implement it.

Object 2 is *ipAdEntIfIndex* and is the second subordinate object type of *ipAddrEntry*. It identifies the instance of occurrence of the entry in the table. It references the values of other elements associated with the interface for that entry occurrence. Although a single element is adequate to uniquely identify the occurrence of an entry in this table, we will see later that there could be more than one element needed in other tables. The syntax of *ipAdEntIfIndex* is INTEGER, a primitive data type. Access and status are read-only and mandatory.

Objects 3, 4, and 5, *ipAdEntNetMask*, *ipAdEntBcastAddr*, and *ipAdEntReasmMaxSize*, respectively, specify the subnet mask, broadcast address information, and the size of the largest datagram. The definition for each describes what the object is.

Object 6 is the managed object, *ipAddrEntry*, which consists of the subordinate object types of 1 to 5 above. It describes the complete set of information consisting of the five fields needed for an entry in the IP interface address table. The syntax for *ipAddrEntry* is a SEQUENCE data type consisting of the five data types. Each data type is identified with its OBJECT DESCRIPTOR and syntax. Note that the access for *ipAddrEntry* is non-accessible. *ipAddrEntry* is itself a subordinate object type of the managed object, *ipAddrTable*. It is the first (and only) element of *ipAddrTable* and has the OBJECT IDENTIFICATION {*ipAddrTable* 1}.

OBJECT 1

	ipAdEntAddr	{ ipAddrEntry 1 }
Syntax		IpAddress
Definition		"The IP address to which this entry's information pertains"
Access		read-only
Status		mandatory

OBJECT 2

	ipAdEntIfIndex	{ ipAddrEntry 2 }
Syntax		INTEGER
Definition		"The index value which uniquely identifies the interface to which this entry is applicable. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex. "
Access		read-only
Status		mandatory

OBJECT 3

	ipAdEntN etMask	{ ipAddrEntry 3 }
Syntax		IpAddress
Definition		"The subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1 and the host bits set to 0."
Access		read-only
Status		mandatory

OBJECT 4

	ipAdEntBcastAddr	{ ipAddrEntry 4 }
Syntax		INTEGER
Definition		"The value of the least-significant bit in the IP broadcast address used for sending datagrams on the (logical) interface associated with the IP address of this entry. For example, when the Internet standard all-ones broadcast address is used, the value will be 1. This value applies to both the subnet and network broadcasts addresses used by the entity on this (logical) interface. "
Access		read-only
Status		mandatory

Figure 4.19 Specifications for an Aggregate Managed Object: ipAddrTable

```
OBJECT 5
        ipAdEntR  easmMaxSize              { ipAddrEntry 5 }
    Syntax              INTEGER (0..65535)
    Definition          "The size of the largest IP datagram which this entity
                        can reassemble from incoming IP fragmented
                        datagrams received on this interface."
    Access              read-only
    Status              mandatory

OBJECT 6
        ipA ddrEntry          { ipAddrTable 1 }
    Syntax              ipAddrEntry ::= SEQUENCE  {
        ipAdEntAddr                 IpAddress,
        ipAdEntIfIndex              INTEGER,
        IpAdEntNetMask              IpAddress,
        IpAdEntBcastAddr            INTEGER,
        ipAdEntReasmMaxSize         INTEGER (0..65535)
    Definition          "The addressing information for one of this entitys'IP
                        addresses."
    Access              not-accessible
    Status              mandatory

OBJECT 7
        ipAddrTable           { ip 20 }
    Syntax              SEQUENCE OF IpAddrEntry
    Definition          "The table of addressing information relevant to this
                        entity's IP addresses."
    Access              not-accessible
    Status              m andatory
```

Figure 4.19 (continued)

ipAddrTable is the OBJECT DESCRIPTOR for the IP address table, which has a unique place in the MIB tree with the OBJECT IDENTIFIER {ip 20}. We will see how the managed object *ip* group fits in the MIB tree in the next section. The syntax of *ipAddrTable* is the structure SEQUENCE OF the data type *ipAddrEntry*. Again, the access is not-accessible.

As an example of the use of the above specifications in a table, let us consider the following entry in an IP address table:

```
OBJECT 1    {ipAdEntAddr } = { internet "123.45.2.1".}
OBJECT 2    {ipAdEntIfIndex} = { "1" }
OBJECT 3    {ipAdEntNetMask} = { internet "255.255.255.0" }
OBJECT 4    {ipAdEntBcastAddr} = { "0" }
OBJECT 5    {ipAdEntReasmMaxSize} = { "12000" }
```

The value of *ipAdEntIfIndex* for this entry in the IP address table is equal to 1, and the IP address defining this interface is 123.45.2.1 using the Internet-specific protocol. The value associated with network mask is 255.255.255.0, with *ipAdEntBcastAddr* 0, and with the maximum size of the packet 12,000.

Figure 4.20 [RFC 1213] presents the macro for the IP address table, a multiple-instance presented in Figure 4.17. The text following "--" are comments and not encoded. The module starts at the highest level defining the *ipAddrTable*, then follows up with *ipAddrEntry* and finally defines the subordinate object types of *ipAddrEntry*. Note that there is an additional clause, INDEX, in the *ipAddrEntry* macro in Figure 4.20. This uniquely identifies the instantiation of the entry object type in the table. Thus,

```
-- the IP address table
-- The IP address table contains this entity's IP addressing information.

        ipAddrTable OBJECT-TYPE
           SYNTAX  SEQUENCE OF IpAddrEntry
           ACCESS  not-accessible
           STATUS   mandatory
           DESCRIPTION
                 "The table of addressing information relevant to this entity's IP addresses."
           ::= { ip 20 }

        ipAddrEntry OBJECT-TYPE
           SYNTAX  IpAddrEntry
           ACCESS  not-accessible
           STATUS  mandatory
           DESCRIPTION
                 "The addressing information for one of this entity's IP addresses."

           INDEX   { ipAdEntAddr }
           ::= { ipAddrTable 1 }

        IpAddrEntry ::=
           SEQUENCE {
              ipAdEntAddr
                 IpAddress,
              ipAdEntIfIndex
                 INTEGER,
              ipAdEntNetMask
                 IpAddress,
              ipAdEntBcastAddr
                 INTEGER,
              ipAdEntReasmMaxSize
                 INTEGER (0..65535) }

        ipAdEntAddr OBJECT-TYPE
           SYNTAX  IpAddress
           ACCESS  read-only
           STATUS  mandatory
           DESCRIPTION
                 "The IP address to which this entry's addressing information pertains."
```

Figure 4.20 Aggregate Managed Object Macro: ipAddrTable [RFC 1155]

```
            ::= { ipAddrEntry 1 }
            ipAdEntIfIndex OBJECT-TYPE
            SYNTAX  INTEGER
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
        "The index value which uniquely identifies the interface to which this entry is applicable. The
interface identified by a particular value of this index is the same interface as identified by
the same value of ifIndex."
            ::= { ipAddrEntry 2 }

            ipAdEntNetMask OBJECT-TYPE
            SYNTAX  IpAddress
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
          "The subnet mask associated with the IP address of this entry. The value of the mask is an
IP address with all the network bits set to 1 and all the host bits set to 0."
            ::= { ipAddrEntry 3 }

            ipAdEntBcastAddr OBJECT-TYPE
            SYNTAX  INTEGER
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
        "The value of the least-significant bit in the IP broadcast address used for sending datagrams
on the (logical) interface associated with the IP address of this entry. For example, when the Internet
standard all-ones broadcast address is used, the value will be 1. This value applies to both
the subnet and network broadcasts addresses used by the entity on this (logical) interface."
            ::= { ipAddrEntry 4 }

            ipAdEntReasmMaxSize OBJECT-TYPE
            SYNTAX  INTEGER (0..65535)
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
        "The size of the largest IP datagram which this entity can reassemble from incoming IP frag-
mented datagrams received on this interface."
        ::= { ipAddrEntry 5 }
```

Figure 4.20 (continued)

ipAdEntAddr object uniquely identifies the instantiation. We will discuss this more in the next section on columnar objects.

We have so far presented the SMI as it was originally developed in RFC 1155. This helped us understand the two aspects of an object module: specifications and formal structure. Obviously, there is duplication in this. It was originally developed this way to eventually migrate to OSI specifications. However, with the reality that the OSI standards were not implemented and SNMP standards had been deployed extensively, the specifications and formal structure were combined into a concise definition of object macro, described in RFC 1212. It is presented in Figure 4.21.

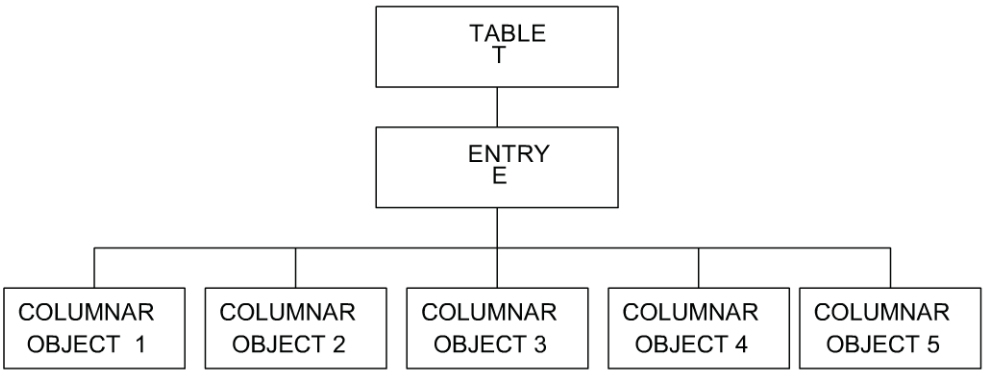
```
IMPORTS
        ObjectName FROM RFC1155-SMI
        DisplayString FROM RFC1158-MIB

OBJECT-TYPE MACRO ::=
BEGIN
        TYPE NOTATION ::=
                                -- must conform to RFC 1155's ObjectSyntax
                "SYNTAX" type    (TYPE ObjectSyntax)
                "ACCESS" Access
                "STATUS" Status
                DescrPart
                ReferPart
                IndexPart
                DefValPart
        VALUE NOTATION ::= value (VALUE ObjectName)

        Access ::=  "read-only" | "read-write" | "write-only" | "not-accessible"
        Status ::= "mandatory" | "optional" | "obsolete" | "deprecated"
        DescrPart ::= "DESCRIPTION" value (description DisplayString) | empty
        ReferPart ::= "REFERENCE" value (reference DisplayString) | empty
        IndexPart ::= "INDEX" "{" IndexTypes "}" | empty
        IndexTypes ::= IndexType | IndexTypes "," IndexType
        IndexType ::=
                        --if indexobject, use SYNTAX
                        --value of the correspondent
                        --OBJECT-TYPE invocation
                value (indexobject ObjectName)
                        --otherwise use named SMI type
                        -- must conform to IndexSyntax below
                        | (type IndexType)
        DefValPart ::=
                "DEFVAL" "{" value (defvalue ObjectSyntax) "}" | empty
END

IndexSyntax ::=
        CHOICE {
                number  INTEGER (0..MAX),
                string          OCTET STRING,
                object          OBJECT IDENTIFIER,
                address         NetworkAddress,
                ipAddress       IpAddress
        }
```

Figure 4.21 OBJECT-TYPE Macro: Concise Definition [RFC 1212]

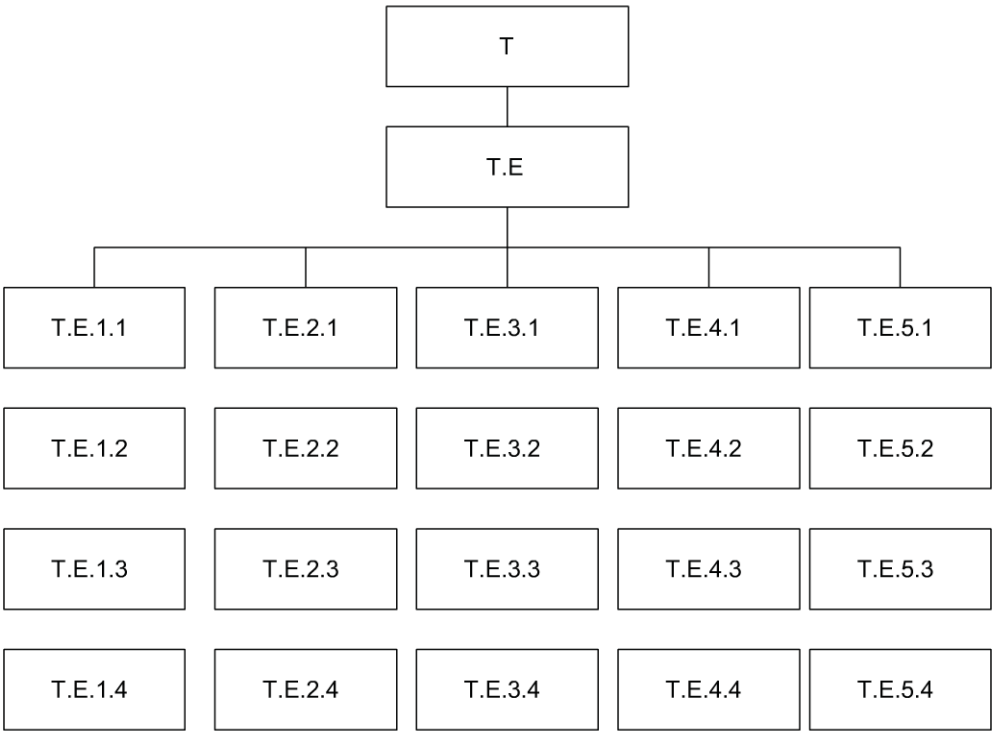
Note that there is the definition of imports from other modules. Also, there are additional clauses, ReferPart, IndexPart, and DefVal, and their associated value definitions. The REFERENCE clause is a textual reference to the document from which the object is being mapped. The INDEX clause is the columnar object identifier, which as we said, will be discussed in the next section under columnar objects. DEFVAL is the default value to the object, if applicable.

Aggregate Object as Columnar Object. The aggregate object that was discussed above has been formally defined as columnar objects in RFC 1212. SNMP operations apply exclusively to scalar operations. This means that a single scalar value is retrieved or edited on a managed object with any one operation. However, managed objects do have multiple instances within a system and need to be represented formally. An aggregate object type comprises one or more subtypes; and each subtype could have multiple instances, with a value associated with each instance.

It is convenient to conceptually define a tabular structure for objects that have multiple values, such as the IP address table. Such tables can have any number of rows including none, with each row containing one or more scalar objects. This is shown in Figure 4.22(a). Table T contains subordinate object Entry E



(a) Multiple-Instance Managed Object



(b) Example of a 5-Columnar Object with 4 Instances (Rows)

Figure 4.22 Numbering Convention of a Managed Object Table

that is a row in the table. Since the table is a SEQUENCE OF construction with entry E as components, there are multiple entries in the table; i.e., there are multiple rows in the table. Entry E is a SEQUENCE construct consisting of subordinate objects, columnar objects 1 through 5, in Figure 4.22(a).

Figure 4.22(b) shows a five-columnar object with four instances, i.e., four rows. It is important to note the convention used in denoting each object in the rows. The columnar objects in each row are denoted by the concatenation of the object identifier of the table, the entry, and then the object, and lastly by the row number. Note that the last two numbers are not like what we would normally think of as a row and column sequence in a matrix representation. It is more like column and row designation. Thus, the third occurrence (third row) of the fourth columnar object (fourth column) is T.E.4.3. The value for the row number is the value of the index of the table. For example, *ipAdEntAddr*, which is the IP address, is the index for the IP address table example shown in Figure 4.20. Hence, the value of *ipAdEntAddr* will determine the row of the table.

Let us apply this conceptual table to the IP address table example we have been following. This is shown in Figure 4.23. Figure 4.23(a) presents the detail of the columnar object, *ipAdEntBcastAddr*, which is the fourth columnar object under *ipAddrEntry*, which is a subordinate object of *ipAddrTable*. The OBJECT IDENTIFIER of the *ipAddrTable* in the MIB is 1.3.6.1.2.1.4.20. The *ipAddrEntry* is node 1 under it and *ipAdEntBcastAddr* is the fourth node under *ipAddrEntry*. Thus, the columnar object identifier of *ipAdEntBcastAddr* is {1.3.6.1.2.1.4.20.1.4}.

Figure 4.23(b) shows the tabular presentation of an IP address table. The table shows four rows and six columns. Each of the four rows in the IP address table indicates a set of values associated with each instance of *ifAddrEntry* in the table.

The first column in Figure 4.23(b) is the row number, which is added to the other five columns (column 2 through 6) that represent the five columnar objects of the IP address table. We have added the first column of the row number for easy explanation only; it is not part of the managed objects. The first columnar object *ipAdEntAddr* is in bold letters to indicate that it is the index for the table. As each row in an aggregate object table is uniquely identified by the INDEX clause of the OBJECT-TYPE macro, each row in our example is uniquely identified by indexing the value of *ipAdEntAddr*. The second row is the columnar object *ipAdEntIfIndex*. Note that *ipAdEntIfIndex*, which is the same as the *ifNumber* of the Interfaces group, is not an index, but just an object associated with each row of the table. The last three columns in Figure 4.23(b) represent the columnar objects *ipAdEntNetMask*, *ipAdEntBcastAddr*, and *ipAdEntReasmMaxSize*.

Figure 4.23(c) shows the representation of the object identifier associated with each instance. There are four instances illustrated in the figure. The first column is the columnar object identifier, the second column is the row number shown in Figure 4.23(b), and the last column is the object identifier for the instance of the columnar object. Let us first look at the first row of Figure 4.23(c). We want to represent the object identifier associated with the columnar object *idAdEntAddr* for the specific occurrence presented in the second row of Figure 4.23(b). The object identifier *ipAdEntAddr* in the first row of Figure 4.23(c) is its columnar object identifier 1.3.6.1.2.1.4.20.1.1. It is suffixed with the value of the table index field *ipAdEntAddr* 123.45.3.4. The resultant object identifier 1.3.6.1.2.1.4.20.1.1.123.45.3.4 is shown in the first row of the last column of Figure 4.23(c).

The second entry in Figure 4.23(c) illustrates the object identifier 1.3.6.1.2.1.4.20.1.2.165.8.9.25 for the columnar object *ipAdEntIfIndex* for the instance indicated in the third row of Figure 4.23(b). The third and fourth entries in Figure 4.23(c) illustrate the object identifier values of *ipAdEntBcastAddr* and *ipAdEntReasmMaxSize* for rows 1 and 4 of Figure 4.23(b), respectively.

```
ipAddrTable {1.3.6.1.2.1.4.20}
        ipAddrEntry (1)
                ipAdEntAddr (1)
                ipAdEntIfIndex (2)
                ipAdEntNetMask (3)
                ipAdEntBcastAddr (4)
                ipAdEntReasmMaxSize (5)

Columnar object ID of  ipAdEntBcastAddr is (1.3.6.1.2.1.4.20.1.4):

iso org dod internet mgmt mib ip ipAddrTable  ipAddrEntry ipAdEntBcastAddr
 1   3   6        1        2    1   4        20            1                  4
```

(a) Columnar Objects under ipAddrEntry

Row	**ipAdEntAddr**	ipAdEntIfIndex	IpAdEntNetMask	IpAdEntBcast Addr	IpAdEntReasmMax Size
1	**123.45.2.1**	1	255.255.255.0	0	12000
2	**123.45.3.4**	3	255.255.0.0	1	12000
3	**165.8.9.25**	2	255.255.255.0	0	10000
4	**9.96.8.138**	4	255.255.255.0	0	15000

(b) Object Instances of ipAddrTable (1.3.6.1.2.1.4.20)

Columnar Object	Row # in (b)	Object Identifier
ipAdEntAddr 1.3.6.1.2.1.4.20.1.1	2	{1.3.6.1.2.1.4.20.1.1.123.45.3.4}
ipAdEntIfIndex 1.3.6.1.2.1.4.20.1.2	3	{1.3.6.1.2.1.4.20.1.2.165.8.9.25}
ipAdEntBcastAddr 1.3.6.1.2.1.4.20.1.4	1	{1.3.6.1.2.1.4.20.1.4.123.45.2.1}
IpAdEntReasmMaxSize 1.3.6.1.2.1.4.20.1.5	4	{1.3.6.1.2.1.4.20.1.5.9.96.8.138}

(c) Object ID for Specific Instances

Figure 4.23 Multiple-Instance Managed Object: ipAddrTable

The formalized definitions of SMI as presented in STD 16/RFC 1155 are shown in Figure 4.24. In addition to the definition of the object type macro, it also specifies the exports of names and object types, as well as the Internet MIB, which is addressed in the next section.

```
RFC1155-SMI DEFINITIONS ::= BEGIN

        EXPORTS -- EVERYTHING
              internet, directory, mgmt, experimental, private, enterprises,
              OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
              ApplicationSyntax, NetworkAddress, IpAddress, Counter, Gauge,
              TimeTicks, Opaque;

        -- the path to the root

        internet          OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }

        directory         OBJECT IDENTIFIER ::= { internet 1 }
        mgmt              OBJECT IDENTIFIER ::= { internet 2 }
        experimental      OBJECT IDENTIFIER ::= { internet 3 }
        private           OBJECT IDENTIFIER ::= { internet 4 }

        enterprises       OBJECT IDENTIFIER ::= { private 1 }

        -- definition of object types

        OBJECT-TYPE MACRO ::=
BEGIN
           TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                             "ACCESS" Access
                             "STATUS" Status
           VALUE NOTATION ::= value (VALUE ObjectName)

           Access ::= "read-only" | "read-write" | "write-only" | "not-accessible"
           Status ::= "mandatory" | "optional" | "obsolete"
END

           -- names of objects in the MIB

           ObjectName ::=
              OBJECT IDENTIFIER

           -- syntax of objects in the MIB

           ObjectSyntax ::=
              CHOICE {
                 simple
                    SimpleSyntax,

           -- Note that simple SEQUENCEs are not directly mentioned here to keep things
           simple (i.e., prevent misuse). However, application-wide types, which are IMPLIC-
           ITly encoded simple SEQUENCEs, may appear in the following CHOICE.

                 application-wide
                    ApplicationSyntax
              }
```

Figure 4.24 SMI Definitions [RFC 1155]

```
SimpleSyntax ::=
   CHOICE {
      number
         INTEGER,
      string
         OCTET STRING,
      object
         OBJECT IDENTIFIER,
      empty
         NULL
   }

ApplicationSyntax ::=
   CHOICE {
      address
         NetworkAddress,
      counter
         Counter,
      gauge
         Gauge,
      ticks
         TimeTicks,
      arbitrary
         Opaque

-- Other application-wide types, as they are defined, will be added here.
   }

-- application-wide types

NetworkAddress ::=
   CHOICE {
      internet
         IpAddress
   }
IpAddress ::=
   [APPLICATION 0]          -- in network-byte order
      IMPLICIT OCTET STRING (SIZE (4))
Counter ::=
   [APPLICATION 1]
      IMPLICIT INTEGER (0..4294967295)
Gauge ::=
   [APPLICATION 2]
      IMPLICIT INTEGER (0..4294967295)
TimeTicks ::=
   [APPLICATION 3]
      IMPLICIT INTEGER (0..4294967295)
Opaque ::=
   [APPLICATION 4]          -- arbitrary ASN.1 value,
      IMPLICIT OCTET STRING   --   "double-wrapped"
END
```

Figure 4.24 (continued)

4.7.4 Management of Information Base

As stated in Section 4.7.1, MIB-II specified in RFC 1213 is the current standard, STD 17. It is a superset of MIB-I or simply MIB, as it was then addressed in RFC 1156. We will present here MIB-II information. Both MIB-I and MIB-II can be implemented in SNMPv1. MIB is organized such that implementation can be done on an as-needed basis. The entire MIB does not have to be implemented in either the manager or the agent process.

Let us remember that MIB is a virtual information store (base). Managed objects are accessed via this virtual information base. Objects in the MIB are defined using ASN.1. In the previous section, we discussed the SMI, which defines the mechanism for describing these objects. The definition consists of three components: *name* (OBJECT DESCRIPTOR), *syntax* (ASN.1), and *encoding* (BER).

Objects defined in MIB-II have the OBJECT IDENTIFIER prefix:

```
mib-2          OBJECT IDENTIFIER ::= {mgmt 1}
```

MIB-II has an additional attribute to the status of a managed object. The new term is "deprecated." This term mandates the implementation of the object in the current version of MIB-II, but is most likely to be removed in future versions. For example, *atTable* is deprecated in MIB-II.

Object Groups. Objects that are related are grouped into object groups. Notice that this grouping is different from the grouping of object types to construct an aggregate object type. Object groups facilitate logical assignment of object identifiers. One of the criteria for choosing objects to be included in standards is that it is essential for either fault or configuration management. Thus, if a group is implemented in a system by a vendor, all the components are implemented, i.e., status is mandatory for all its components. For example, if the External Gateway Protocol (EGP) is implemented in a system, then all EGP group objects are mandatory to be present.

The MIB module structure consists of the module name, imports from other modules, and definitions of the current module. The basic ASN.1 structure is shown in Figure 4.25.

There are 11 groups defined in MIB-II. The tree structure is shown in Figure 4.26, and Table 4.4 presents the name, object identification (OID), and a brief description of each group. It can be observed that these groups are nodes under the MIB object *mib-2* whose OBJECT IDENTIFIER is 1.3.6.1.2.1.

The System group contains objects describing system administration. The Interfaces group defines interfaces of the network component and network parameters associated with each of those interfaces. The Address translation group is a cross-reference table between the IP address and the physical address. IP, ICMP, TCP, UDP, and EGP groups are the grouping of objects associated with the respective protocol of the system. The group, CMOT, is a placeholder for future use of the OSI protocol, CMIP over TCP/IP. The Transmission group was created as a placeholder for network transmission-related parameters and was a placeholder in RFC 1213. Numerous transmission systems and objects have been developed under this group since then. SNMP group is the communication protocol group associated with SNMP

```
<module name > DEFINITIONS ::= BEGIN
        <imports>
        <definitions>
END
```

Figure 4.25 MIB Module Structure

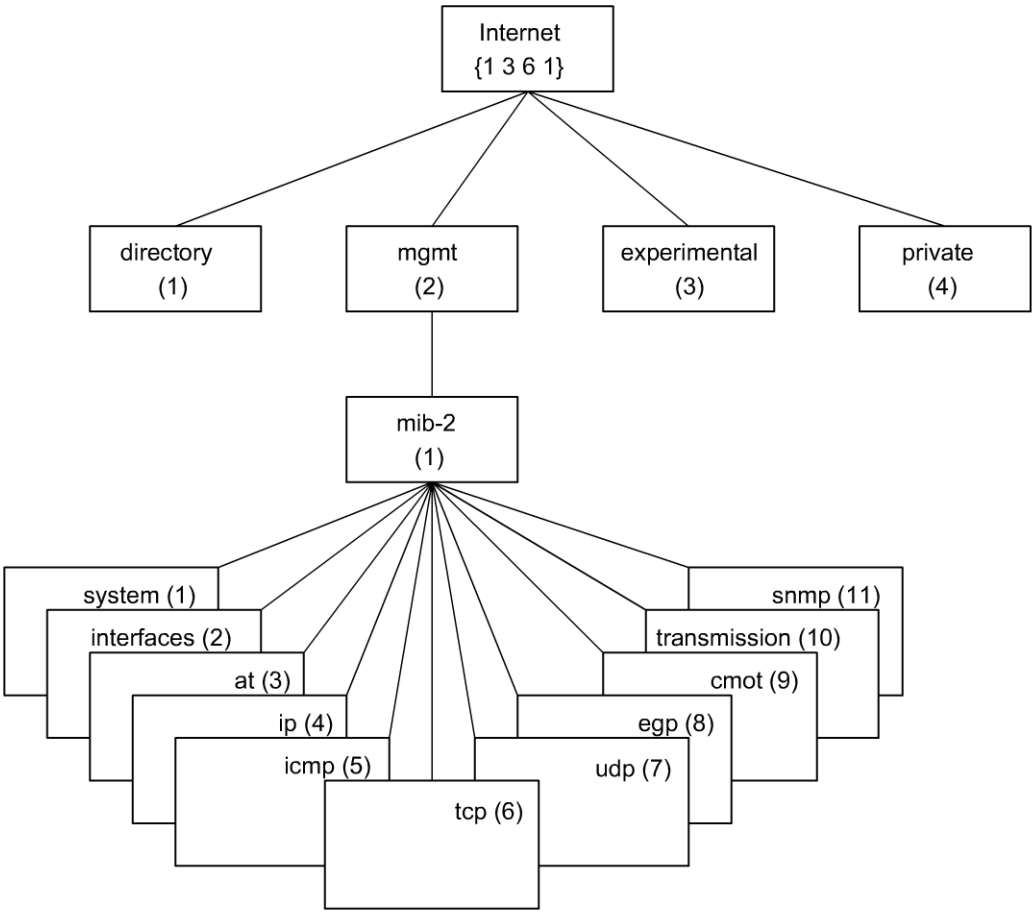


Figure 4.26 Internet MIB-II Group

Table 4.4 MIB-II Groups

GROUP	OID	DESCRIPTION (BRIEF)
system	mib-2 1	System description and administrative information
interfaces	mib-2 2	Interfaces of the entity and associated information
at	mib-2 3	Address translation between IP and physical address
ip	mib-2 4	Information on IP protocol
icmp	mib-2 5	Information on ICMP protocol
tcp	mib-2 6	Information on TCP protocol
udp	mib-2 7	Information on UDP protocol
egp	mib-2 8	Information on EGP protocol
cmot	mib-2 9	Placeholder for OSI protocol
transmission	mib-2 10	Placeholder for transmission information
snmp	mib-2 11	Information on SNMP protocol

management. We will now learn more about some of these groups. It should be noted that there are more groups defined under the Internet node, which we will address in Chapter 5.

The following sections describe details of each group except for CMOT, transmission, and SNMP. The CMOT group is a placeholder and is not yet defined. The Transmission group is based on the transmission media underlying each interface of the system; the corresponding portion of the Transmission group is mandatory for that system. The SNMP group will be addressed in Chapter 5 as part of the communication model.

Although there are many more groups in MIB-II, details on only the generic groups directly related to physical properties of basic network elements (System and Interfaces) and the managed objects associated with Internet protocols (IP, TCP, and UDP) are presented here. They are intended to familiarize the reader quickly with how to read and interpret RFCs specifying MIBs. It is strongly recommended that you refer to the RFC for detailed specifications on each group and understand the structure of each MIB group.

Some examples associated with managed objects in the group are presented along with a description of the group in order to appreciate the significance of each MIB. In Chapter 9 we will learn to use the SNMP command using SNMP tools and retrieve the values associated with managed objects.

System Group. The System group is the basic group in the Internet standard MIB. Its elements are probably the most accessed managed objects. After an NMS discovers all the components in a network or newly added components in the network, it has to obtain information on the system it discovered, such as system name, object ID, etc. The NMS will initiate the get-request command on the objects in this group for this purpose. Data on the systems shown in Figure 4.2 were obtained by the NMS using this group. The group also has administrative information, such as contact person and physical location that helps a network manager.

Implementation of the System group is mandatory for all systems in both the agent and the manager. It consists of seven entities, which are presented in Figure 4.27 and Table 4.5. The vendor of the equipment programs the system description (*sysDescr*) and OBJECT IDENTIFIER (*sysObjID*) during manufacturing. System up time is filled in hundredths of a second dynamically during operation. Network management systems usually convert this into a readable format of days, hours, and minutes in their presentation, as shown in Figure 4.2. Although system services (*sysServices*) object is mandatory to be implemented, most NMSs do not show the information automatically.

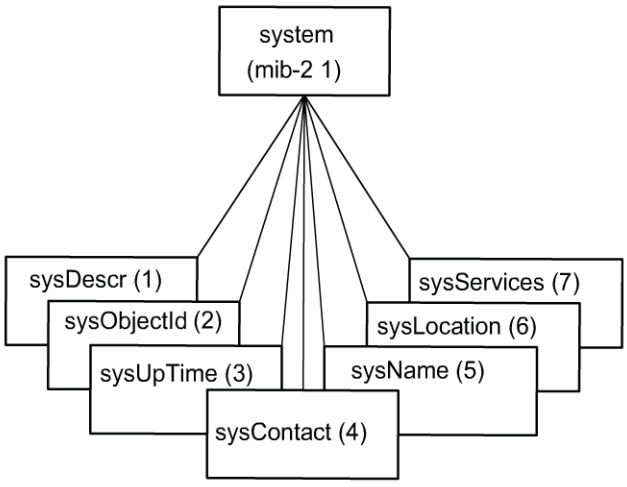


Figure 4.27 System Group

Table 4.5 System Group

ENTITY	OID	DESCRIPTION (BRIEF)
sysDescr	system 1	Textual description
sysObjectID	system 2	OBJECT IDENTIFIER of the entity
sysUpTime	system 3	Time (in hundredths of a second since last reset)
sysContact	system 4	Contact person for the node
sysName	system 5	Administrative name of the system
sysLocation	system 6	Physical location of the node
sysServices	system 7	Value designating the layer services provided by the entity

Interfaces Group. The Interfaces group contains managed objects associated with the interfaces of a system. If there is more than one interface in the system, the group describes the parameters associated with each interface. For example, if an Ethernet bridge has several network interface cards, the group would cover information associated with each interface. However, the Interfaces MIB contains only generic parameters. In the Ethernet example, there is more information associated with the Ethernet LAN, which is addressed in the MIB specifications of the particular medium, as in Definitions of Managed Objects for the Ethernet-like Interface types [RFC 2358]. An NMS would combine information obtained from various groups in presenting comprehensive data to the user.

The Interfaces group specifies the number of interfaces in a network component and managed objects associated with each interface. Implementation of Interfaces group is mandatory for all systems. It consists of two nodes as shown in Figure 4.28 and Table 4.6. The number of interfaces of the entity is defined by *ifNumber*, and the information related to each interface is defined in the Interfaces table, *ifTable*.

Each interface in the Interfaces table can be visualized as being attached to either a subnetwork or a system. The term **subnetwork** is not to be confused with the term **subnet**, which refers to an addressing partitioning scheme in the Internet suite of protocols. The index for the table is just one entity, specified by *ifIndex*, as shown below in the definition of the *ifEntry* module under *ifTable*.

```
IfEntry  OBJECT-TYPE
         SYNTAX        ifEntry
         ACCESS        not-accessible
         STATUS        mandatory
         DESCRIPTION
                 "An interface entry containing objects at the subnetwork layer and below for a
                 particular interface."
         INDEX  {ifIndex}
         ::= {ifTable 1}
```

The index is also shown in bold letters in the figure and the table.

The entity *ifType* describes the type of data link layer directly below the network layer. It is defined as an enumerated integer. Examples of these are: ethernet-csmacd(7), iso88025-tokenRing(9). See RFC 1213 for the specified type of standard interfaces.

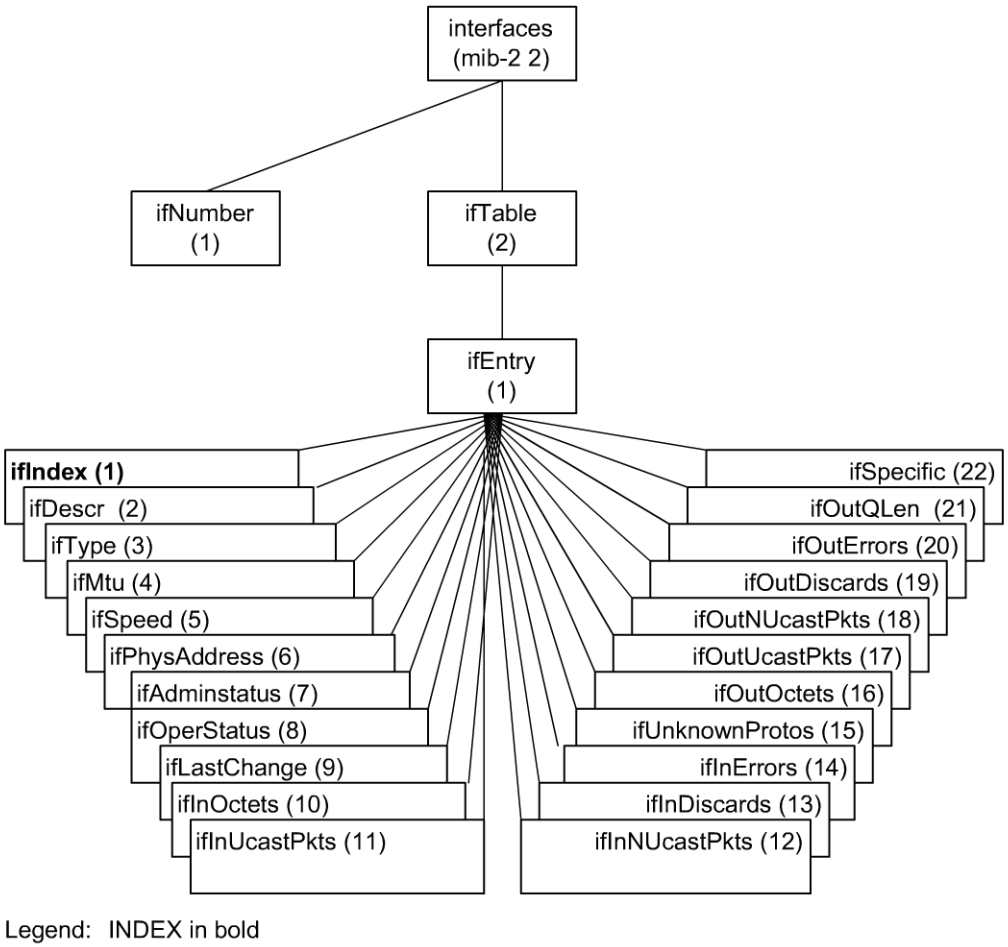


Figure 4.28 Interfaces Group

The administrative and operational status that is indicated by object identifiers 7 and 8 should agree with each other when the system interface is functioning as administered.

Object identifiers 11–15 refer to the measurements (with counter syntax) on inbound traffic and object identifiers 16–21 to measurements on outbound traffic.

An example of use of Interfaces MIB would be to measure the incoming and the outgoing traffic rate on a given interface of an Ethernet hub. We can specify a port on an Ethernet network interface card by the value of *ifIndex* and query (*get-request*) the number of input unicast packets (*ifInUcastPkts*) and the number of output unicast packets (*ifOutUcastPkts*) every second. Remember that we get the reading of two counters, which are incremented with every packet coming in or going out of the port from the management agent associated with the port. We would then take the difference in the consecutive counter reading to derive the packet rate of traffic with time.

Interface Sublayers. One of the strengths of an IP network layer protocol is that it is designed to run over any network interface. IP considers any and all protocols it runs over as a single "network interface" layer. The Interfaces group defines a generic set of managed objects such that any network interface can be managed in an interface-independent manner through these managed objects. The Interfaces group provides the means for additional managed objects specific to particular types of network interface (e.g., a specific medium such as Ethernet or Time Division Multiplex (TDM) channels) to be

Table 4.6 Interfaces Group

ENTITY	OID	DESCRIPTION (BRIEF)
ifNumber	interfaces 1	Total number of network interfaces in the system
ifTable	interfaces 2	List of entries describing information on each interface of the system
ifEntry	ifTable 1	An interface entry containing objects at the subnetwork layer for a particular interface
ifIndex	ifEntry 1	A unique integer value for each interface
ifDescr	ifEntry 2	Textual data on product name and version
ifType	ifEntry 3	Type of interface layer below the network layer defined as an enumerated integer
ifMtu	ifEntry 4	Largest size of the datagram for the interface
ifSpeed	ifEntry 5	Current or nominal data rate for the interface in bps
ifPhysAddress	ifEntry 6	Interface's address at the protocol layer immediately below the network layer
ifAdminStatus	ifEntry 7	Desired status of the interface: up, down, or testing
ifOperStatus	ifEntry 8	Current operational status of the interface
ifLastchange	ifEntry 9	Value of sysUpTime at the current operational status
ifInOctets	ifEntry 10	Total number of input octets received
ifInUcastPkts	ifEntry 11	Number of subnetwork unicast packets delivered to a higher-layer protocol
ifInNUcastPkts	ifEntry 12	Number of non-unicast packets delivered to a higher-layer protocol
ifInDiscards	ifEntry 13	Number of inbound packets discarded irrespective of error status
ifInErrors	ifEntry 14	Number of inbound packets with errors
ifInUnknownProtos	ifEntry 15	Number of unsupported protocol packets discarded
ifOutOctets	ifEntry 16	Number of octets transmitted out of the interface
ifOutUcastPkts	ifEntry 17	Total number of unicast packets that higher-level layer requested to be transmitted
ifOutNUcastPkts	ifEntry 18	Total number of non-unicast packets that higher-level layer requested to be transmitted
ifOutDiscrds	ifEntry 19	Number of outbound packets discarded irrespective of error status
ifOutErrors	ifEntry 20	Number of outbound packets that could not be transmitted because of errors
ifOutQLen	ifEntry 21	Length of the output queue in packets
ifSpecific	ifEntry 22	Reference to MIB definitions specific to the particular media used to realize the interface

defined as extensions to the Interfaces group for media-specific management. Since the standardization of MIB-II, many such media-specific MIB modules have been defined. Concurrently, the Interfaces group has evolved to accommodate the additional managed objects that need to be specified in a data link layer (DLL)—Layer 2.

DLL can be visualized, in general, as comprising several sublayers. These can either be horizontally stacked or vertically sliced (or "stacked"), as shown in Figures 4.29(a) and (b), respectively. An example of the former is an interface with PPP running over a High data rate Digital Subscriber Line (HDLC) link, which uses an RS232-like connector. An example of the latter is a cable access link with a downstream channel and several upstream channels.

Since the simplistic model of a single conceptual row in the *ifTable* in the Interfaces group, an additional MIB group, *ifMIB* (mib-2 31) was created. This is not shown in Figure 4.26, which is the original MIB-II Group. It is shown in Figure 4.30. The first subnode of *ifMIB* is *ifMIBObjects.* There are other subnodes under *ifMIB* and the reader is referred to [RFC 2863] for details. Under the subnode *ifMIBObjects (ifMIB 1)*, there are three tables *ifXTable (ifMIBObjects 1), ifStackTable (ifMIBObjects 2),* and *ifRcvAddressTable (ifMIBObjects 4).* Including the *ifTable (interfaces 2)*, there are four generic Interfaces group tables under the two MIBs, *interfaces* and *ifMIB,* which we should be concerned with in defining managed objects in the DLL layer. In addition to this, there are device-specific interface MIBs, such as Ethernet-like managed objects (*transmission 7*) that we would discuss under each subject as we deal with them. It is worth noting that specifications for *ifMIB* have gone through a series of documentation—RFC 1229, RFC 1573, RFC 2233, and RFC 2863—each obsolescing the previous version.

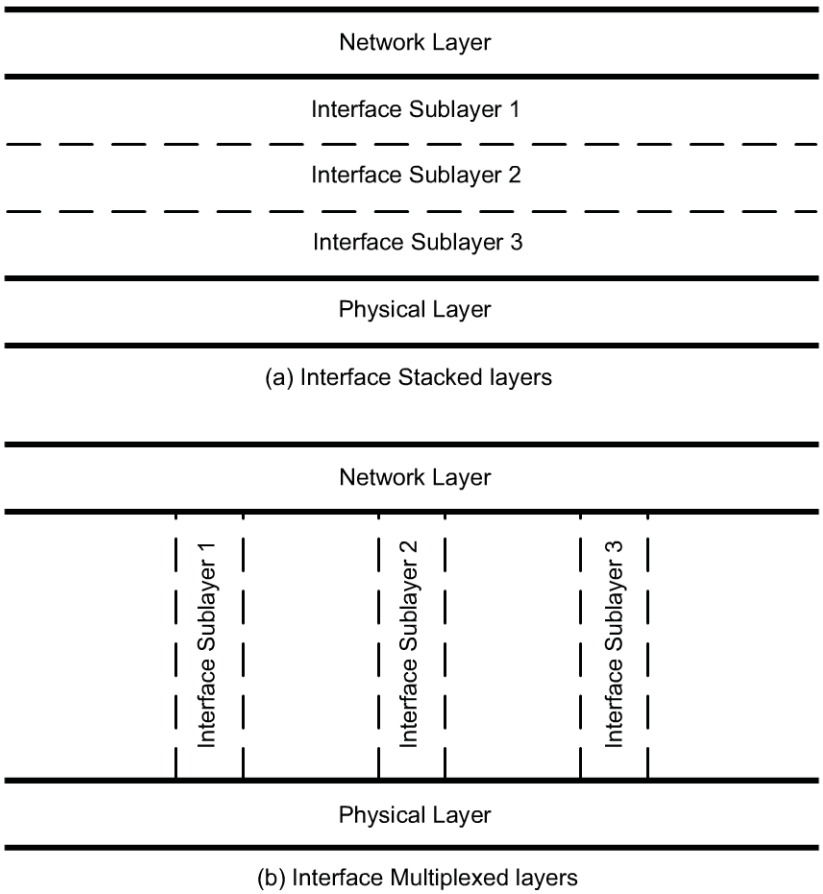


Figure 4.29 Interface Sublayers

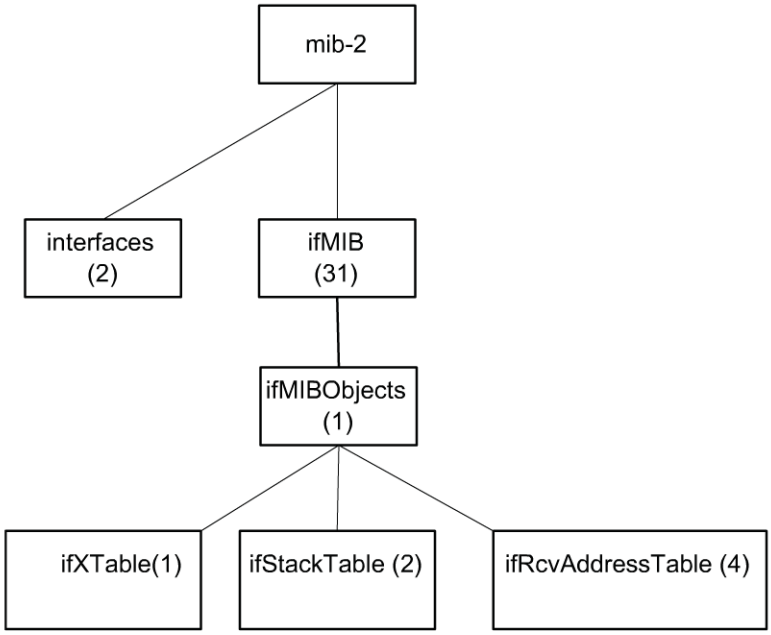


Figure 4.30 Interfaces Groups

ifXTable contains objects that have been added to the Interface MIB group as a result of the Interface evolution effort, or replacements for objects of the original (MIB-II) *ifTable* that were deprecated because the semantics of the said objects have significantly changed. It is an augmentation of *ifTable.* How two tables are augmented in SMI to appear as a single table is described in Chapter 6 under SNMP2.

ifStackTable contains objects that define relationships among sublayers of an interface. Each sublayer is *defined* as an *ifType* and is represented by a conceptual row in the *ifTable*. Because of the addition of such conceptual rows, the value of *ifIndex* is no longer constrained. In other words, it can be stated that the index of a conceptual row no longer has to be less than or equal to the value of *ifIndex*. The upper layer in the *ifStackTable, ifStackHigherLayer,* is the sublayer above the sublayer under consideration and carries the value of the *ifIndex* of that sublayer. If there is no interface sublayer above, i.e., it interfaces directly with the network layer, then the *ifIndex* value is zero. Similarly, *ifStackLowerLayer* is the lower interface sublayer, it has a corresponding *ifIndex* value of that row. If it interfaces directly with the physical medium, its value is zero.

ifRcvAddressTable contains objects that are used to define media-level addresses, which this *interface* will receive, such as a port ID. This table is a generic table.

Address Translation Group. The Address Translation group consists of a table that converts NetworkAddress to a physical or subnetwork address for all interfaces of the system. For example, in Ethernet the translation table is ARP cache. Since in MIB-II each protocol group contains its own translation table, this is not needed and hence its status is deprecated. It is mandatory to be implemented to be backward compatible with MIB-I.

IP Group. The Internet is based on IP protocol as the networking protocol. This group has information on various parameters of the protocol. It also has a table that replaces the Address Translation table. Routers in the network periodically execute the routing algorithm and update its routing table, which are defined as managed objects in this group. We will discuss the contents of this group in detail now.

The IP group defines all the parameters needed for the node to handle a network layer IP protocol either as a host or as a router; implementation is mandatory. Figure 4.31 and Table 4.7 present the tree structure and details of the entities, respectively. The group contains three tables, IP address table, IP routing table, and IP Address Translation table.

We can use the IP MIB to acquire any information associated with the IP layer. For example, to learn the value of the managed object, *ipForwarding* will indicate whether the node is acting as just a router or a gateway between two autonomous networks. We can measure IP datagrams received that are in error, such as those with wrong addresses (*ipInAddrErrors*).

The three tables belonging to the IP group are shown in Figure 4.32 (IP Address Table), Figure 4.33 (IP Routing Table), and Figure 4.34 (IP Address Translation Table). Table 4.8 shows the entity table for the IP address table. The index for the table, *ipAdEntAddr*, is shown in bold letters.

In Figure 4.23(b), we illustrated an example of four instantiations (rows) associated with the IP address table. The IP address table MIB shown in Figure 4.32 and Table 4.8 is used to retrieve data from the router. It could be retrieved using *get-request* or *get-next-request* commands.

The IP routing table is shown in Figure 4.33 and Table 4.9. It contains an entry for each route presently known to the entity. Multiple routes, up to five, to a single destination can appear in the table, but access to such multiple entries is dependent on the table-access mechanism defined by the network management protocol. Routes are indicated by the entities, *ipRouteMetricN*, where *N* is any integer from 1

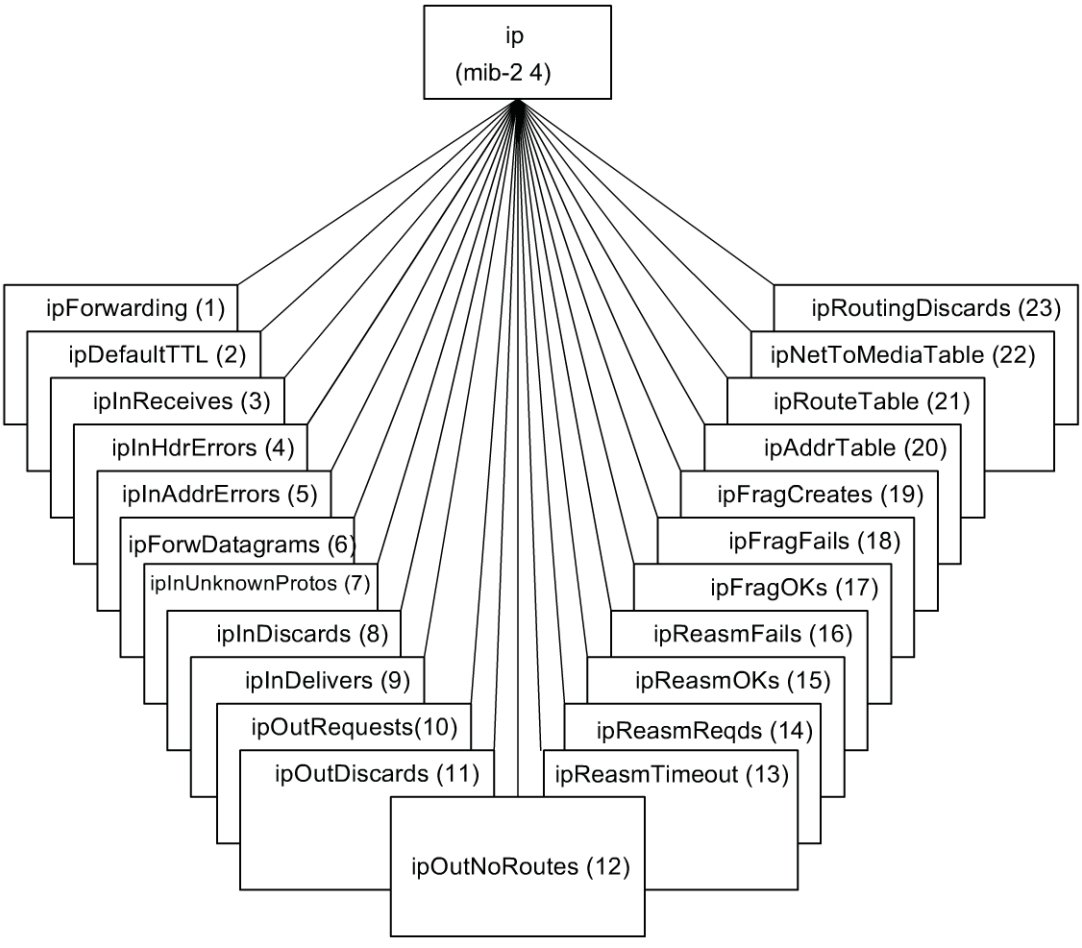
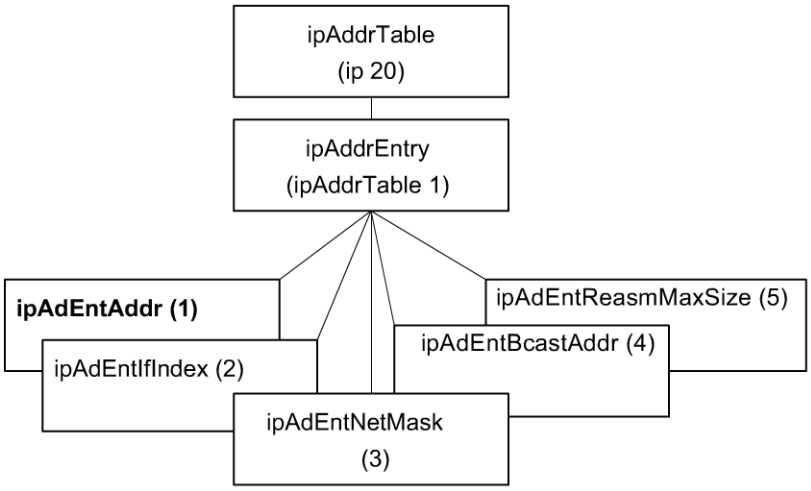


Figure 4.31 IP Group

Table 4.7 IP Group

ENTITY	OID	DESCRIPTION (BRIEF)
ipForwarding	ip 1	Node acting as a gateway or not
ipDefaultTTL	ip 2	Time-to-Live field of the IP header
ipInReceives	ip 3	Total number of input datagrams received from interfaces, including those in error
ipInHdrErrors	ip 4	Number of datagrams discarded due to header errors
ipInAddrErrors	ip 5	Number of datagrams discarded due to address errors
ipForwDatagrams	ip 6	Number of input datagrams attempted to forward to the destination; successfully forwarded datagrams for source routing
ipInUnknownProtos	ip 7	Number of locally addressed datagrams received successfully but discarded due to unsupported protocol
ipInDiscards	ip 8	Number of input datagrams discarded with no problems (e.g. back of buffer space)
ipInDelivers	ip 9	Total number of input datagrams successfully delivered to IP user protocols
ipOutRequests	ip 10	Total number of IP datagrams that local IP user protocols supplied to IP
ipOutDiscards	ip 11	Number of no-error IP datagrams discarded with no problems (e.g. lack of buffer space)
ipOutNoRoutes	ip 12	Number of IP datagrams discarded because no route could be found to transmit them to their destination
ipReasmTimeOut	ip 13	Maximum number of seconds that received fragments are held while they are awaiting reassembly
ipReasmReqds	ip 14	Number of IP datagrams received needing reassembly
ipReasmOKs	ip 15	Number of successfully reassembled datagrams
ipReasmFails	ip 16	Number of failures detected by the IP reassembly algorithm (not discarded fragments)
ipFragOKs	ip 17	Number of successfully fragmented datagrams
ipFragFails	ip 18	Number of IP datagrams not fragmented due to “Don’t Fragment Flag” set
ipFragCreates	ip 19	Number of datagram fragments generated as a result of fragmentation
ipAdddrTable	ip 20	Table of IP addresses
ipRouteTable	ip 21	IP routing table containing an entry
ipNetToMediaTable	ip 22	IP Address Translation table mapping IP addresses to physical addresses
ipRoutingDiscards	ip 23	Number of routing entries discarded even though they were valid



Legend: INDEX in bold

Figure 4.32 IP Address Table

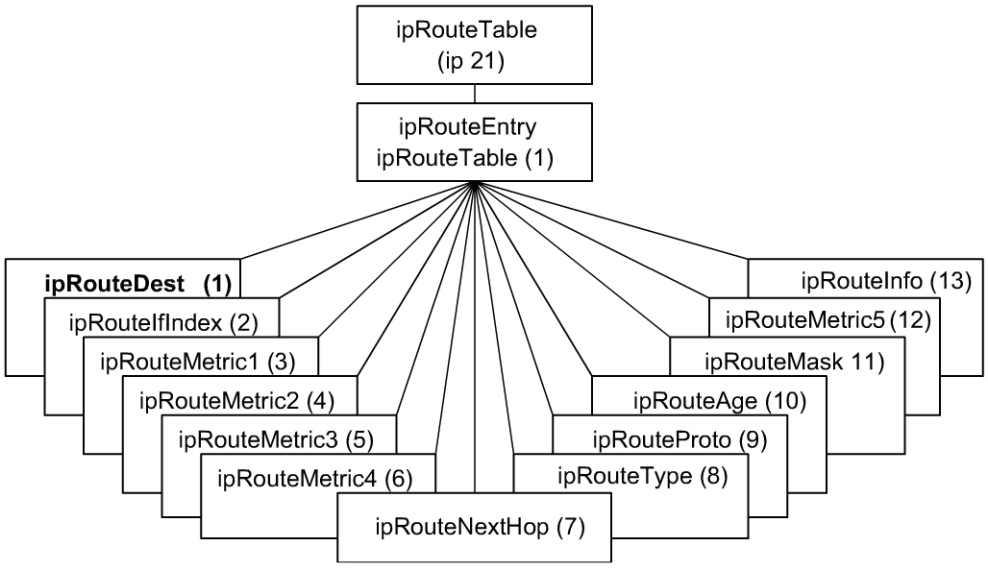


Figure 4.33 IP Routing Table

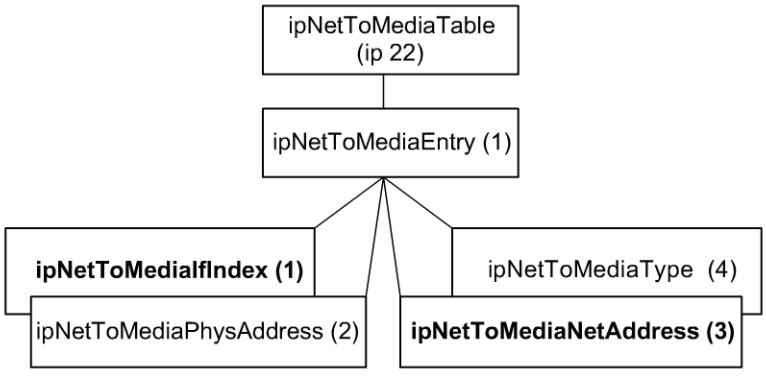


Figure 4.34 IP Address Translation Table

Table 4.11 IP Forwarding Table

ENTITY	OID	DESCRIPTION (BRIEF)
ipForward	ip 24	Contains information on IP forwarding table; deprecates IP routing table
ipForwardNumber	ipForward 1	Number of entries in the IP forward table
ipForwardTable	ipForward 2	Routing table of this entity
ipForwardEntry	IpForwardTable 1	A particular route to a particular destination under a particular policy
ipForwardDest	IpForwardEntry 1	Destination IP route of this address
ipForwardMask	IpForwardEntry 2	Mask to be logically ANDed with the destination address before comparing with the ipRouteDest field
ipForwardPolicy	IpForwardEntry 3	Set of conditions that selects one multipath route
ipForwardNextHop	IpForwardEntry 4	Address of the next system
ipForwardIfIndex	IpForwardEntry 5	ifIndex value of the interface
ipForwardType	IpForwardEntry 6	Type of route: remote, local, invalid, or otherwise; enumerated integer syntax
ipForwardProto	IpForwardEntry 7	Routing mechanism by which this route was learned
ipForwardAge	IpForwardEntry 8	Number of seconds since routing was last updated
ipForwardInfo	IpForwardEntry 9	Reference to MIB definition specific to the routing protocol
ipForwardNextHopAS	IpForwardEntry 10	Autonomous system number of Next Hop
ipForwardMetric1	IpForwardEntry 11	Primary routing metric for this route
ipForwardMetric2	IpForwardEntry 12	An alternative routing metric for this route
ipForwardMetric3	IpForwardEntry 13	An alternative routing metric for this route
ipForwardMetric4	IpForwardEntry 14	An alternative routing metric for this route
ipForwardMetric5	IpForwardEntry 15	An alternative routing metric for this route

The entity *ipForwardPolicy* defines the general set of conditions that would cause the selection of one multipath route over others. Selections of path can be done by the protocol. If it is not done by the protocol, it is then specified by the IP Type-of-Service (TOS) Field, which is a part of the IP type of service field. See Baker [RFC 1354] for more details.

ICMP Group. We used the ICMP to do some of the networking exercises in Chapter 1. It is part of the TCP/IP suite of protocols. All parameters associated with ICMP protocol are covered in this group.

As mentioned in Section 4.2, ICMP is a precursor of SNMP and a part of the TCP/IP suite. It is included in MIB-I and MIB-II and implementation is mandatory. The ICMP group contains statistics on ICMP control messages of ICMP and is presented in Figure 4.36 and Table 4.12. The syntax of all entities is read-only counter. For example, statistics on the number of *ping* requests (icmp echo request) sent might be obtained from the counter reading of *icmpOutEchoes*.

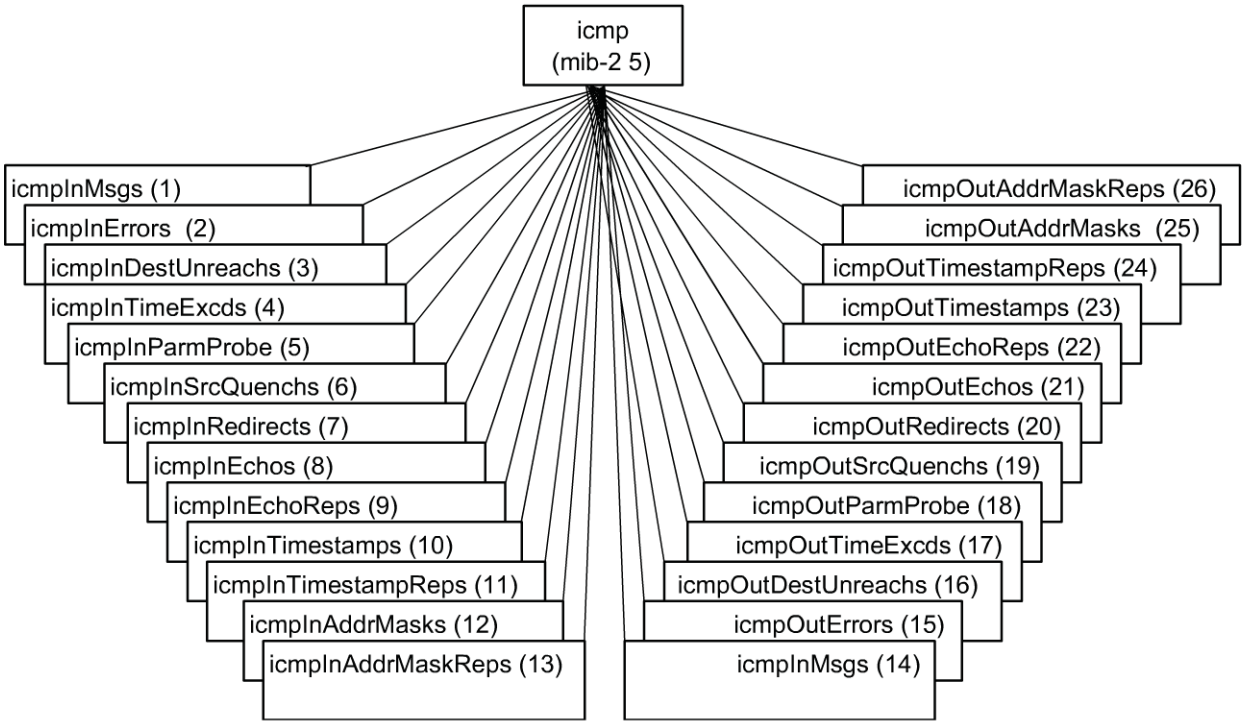


Figure 4.36 ICMP Group

Table 4.12 ICMP Group

ENTITY	OID	DESCRIPTION (BRIEF)
icmpInMsgs	icmp 1	Total number of ICMP messages received by the entity including icmpInErrors
icmpInErrors	icmp 2	Number of messages received by the entity with ICMP-specific errors
icmpInDestUnreachs	icmp 3	Number of ICMP Destination Unreachable messages received
icmpInTimeExcds	icmp 4	Number of ICMP Time Exceeded messages received
icmpInParmProbs	icmp 5	Number of ICMP Parameter Problem messages received
icmpInSrcQuenches	icmp 6	Number of ICMP Source Quench messages received
icmpInRedirects	icmp 7	Number of ICMP Redirect messages received
icmpInEchos	icmp 8	Number of ICMP Echo (request) messages received
icmpInEchoReps	icmp 9	Number of ICMP Echo Reply messages received
icmpInTimestamps	icmp 10	Number of ICMP Timestamp (request) messages received
icmpInTimestampReps	icmp 11	Number of ICMP Timestamp Reply messages received
icmpInAddrMasks	icmp 12	Number of ICMP Address Mask Request messages received

Table 4.12 (continued)

ENTITY	OID	DESCRIPTION (BRIEF)
icmpInAddrMaskReps	icmp 13	Number of ICMP Address Mask Reply messages received
icmpOutMsgs	icmp 14	Total number of ICMP messages attempted to be sent by this entity
icmpOutErrors	icmp 15	Number of good ICMP messages not sent, does not include the ones with errors
icmpOutDestUnreachs	icmp 16	Number of ICMP Destination Unreachable messages sent
icmpOutTimeExcds	icmp 17	Number of ICMP Time Exceeded messages sent
icmpOutParmProbs	icmp 18	Number of ICMP Parameter Problem messages sent
icmpOutSrcQuenchs	icmp 19	Number of ICMP Source Quench messages sent
icmpOutRedirects	icmp 20	Number of ICMP Redirect messages sent
icmpOutEchos	icmp 21	Number of ICMP Echo (request) messages sent
icmpOutEchoReps	icmp 22	Number of ICMP Echo Reply messages sent
icmpOutTimestamps	icmp 23	Number of ICMP Timestamp (request) messages sent
icmpOutTimestampReps	icmp 24	Number of ICMP Timestamp Reply messages sent
icmpOutAddrMasks	icmp 25	Number of ICMP Address Mask Request messages sent
icmpOutAddrMaskReps	icmp 26	Number of ICMP Address Mask Reply messages sent

TCP Group. The transport layer of the Internet defines Transmission Control Protocol (TCP) for a connection-oriented circuit and User Datagram Protocol (UDP) for a connectionless circuit. We will describe the TCP group in this section and UDP in the next subsection.

The TCP group contains entities that are associated with the connection-oriented TCP. They are present only as long as the particular connection persists. It is mandatory to implement this group. The entities are shown in Figure 4.37 and Table 4.13. It contains one table, the TCP connection table, which is presented in Figure 4.38 and Table 4.14. The table entry has four indices to uniquely define it in the table. They are: *tcpConnLocalAddress, tcpConnLocalPort, tcpConnRemAddress*, and *tcpConnRemPort* and are identified in boldface. One may obtain all TCP active sessions from this table with addresses and ports of local and remote entities.

UDP Group. The UDP group contains information associated with the connectionless transport protocol. Its implementation is mandatory. Figure 4.39 and Table 4.15 present the UDP group tree structure and entities, respectively. The group contains a UDP listener table, shown as part of Figure 4.39 and Table 4.15. The table contains information about the entity's UDP end-points on which a local application is currently accepting datagrams. Indices for the table entry are *udpLocalAddress* and *udpLocalPort*, and are indicated in bold letters.

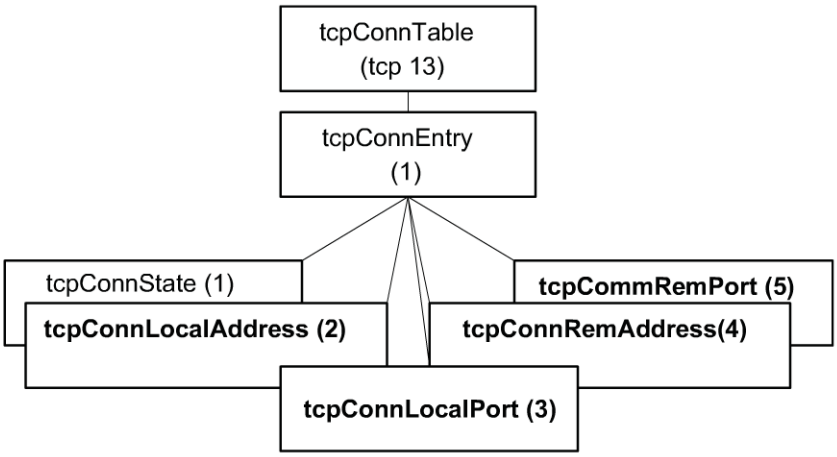


Figure 4.38 TCP Connection Table

Table 4.14 TCP Connection Table

ENTITY	OID	DESCRIPTION (BRIEF)
tcpConnTable	tcp 13	TCO connection table
tcpconnEntry	TcpConnTable 1	Information about a particular TCP connection
tcpConnState	TcpConnEntry 1	State of the TCP connection
tcpConnLocalAddress	TcpConnEntry 2	Local IP address
tcpConnLocalPort	TcpConnEntry 3	Local port number
tcpConnRemAddress	TcpConnEntry 4	Remote IP address
tcpConnRemPort	TcpConnEntry 5	Remote port number

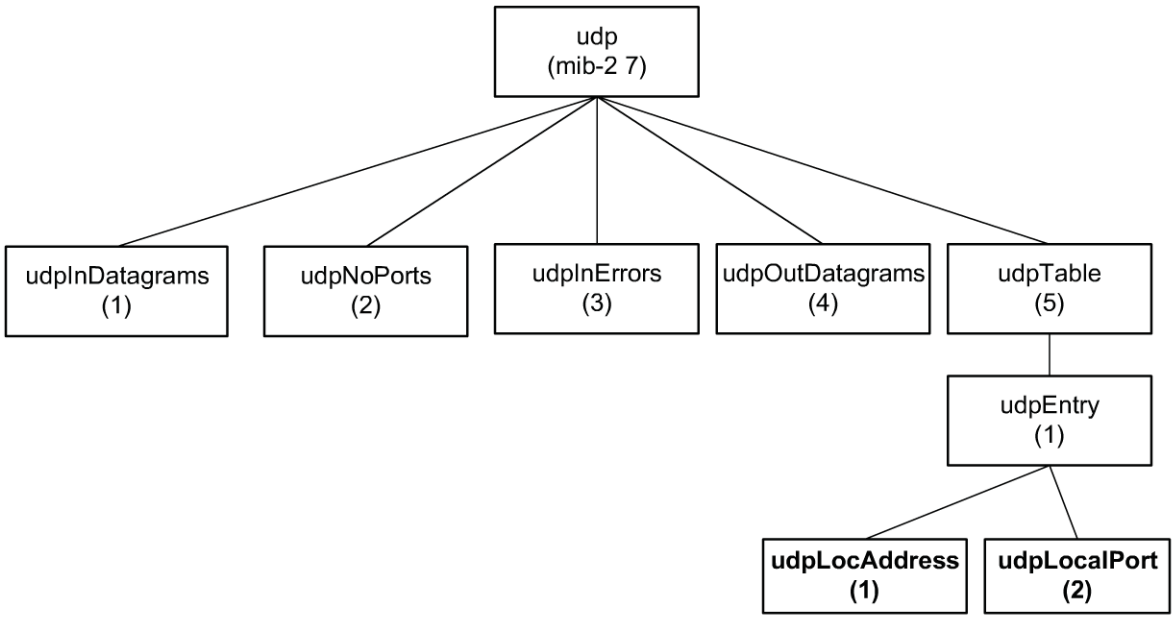


Figure 4.39 UDP Group

Table 4.15 UDP Group

ENTITY	OID	DESCRIPTION (BRIEF)
udpInDatagrams	udp 1	Total number of datagrams delivered to the users
udpNoPorts	udp 2	Total number of received datagrams for which there is no application
udpInErrors	udp 3	Number of received datagrams with errors
udpOutDatagrams	udp 4	Total number of datagrams sent
udpTable	udp 5	UDP Listener table
udpEntry	udpTable 1	Information about a particular connection or UDP listener
udpLocalAddress	udpEntry 1	Local IP address
udpLocalPort	udpEntry 2	Local UDP port

Summary

We have learned the basic functions of SNMP management in this chapter. Advanced functions are covered in the next chapter. The subject matter included in this chapter has been approved as a standard by IETF and implemented by most vendors.

We briefly learned the historical development of SNMP standards and documents. They grew more out of practical necessity than the need for setting standards. The Internet Engineering Task Force is the standards organization and RFC, STD, and FYI are IETF documents on standards development.

SNMP management is organized as two-tier management, in which a manager process and an agent process communicate with each other. The agent process resides in the network element. The manager process is built in network management stations. The agent process does not perform any analysis, which is done in the manager. The two-tier structure can be extended to three-tier by sandwiching a proxy agent, or RMON, between the manager and the agent.

All management operations are done using five messages in SNMPv1, which is the current standard. They are get-request, get-next, set-request, get-response, and trap. The first three are sent from the manager to the agent, and the last two are sent by the agent to the manager.

Messages are exchanged according to specifications defined in the Structure of Management Information (SMI). It is composed of name, syntax, and encoding rules. The name is a unique name for the managed object and an associated unique object identifier. The syntax uses Abstract Syntax Notation 1 (ASN.1) language. Encoding is done using basic encoding rules (BER).

Objects or entities can be composed of other scalar objects. Multiple instances of a managed object, such as the IP address table, are handled by defining tables and columnar objects in the table. Managed objects are organized in a virtual database, called the Management Information Base (MIB). It is distinct from the management database that contains values for managed objects. Managed objects are grouped in the MIB according to their function. MIB-II, which is a superset of MIB-I, consists of 11 groups. Several groups have since been added to the MIB, although they have not been approved as a standard.

Exercises

1. Refer to Figure 4.3 to answer the following questions:
   (a) What are the classes of networks shown in Figure 4.3(a)?
   (b) Explain the function of a network mask.

(c) In Figure 4.3(c), network addresses 172.16x.0 are subnets derived from the network address 172.160.0. Explain how IP address bits are split between subnet and host addresses.

2. Access Simple Gateway Monitoring Protocol (SGMP) RFC 1028 on the Internet. Describe the four message types defined in the document. (You do not have to present the structure of the message.)
3. Present OBJECT IDENTIFIER for the object sun.products in two different formats, one in all mnemonic and the other in all numeric.
4. Represent the objects as OBJECT IDENTIFIERs starting from the root for the three network components in Figure 4.2.
   (a) Hub in Figure 4.2(a) in hybrid format
   (b) Hub in Figure 4.2(b) in numeric format
   (c) Router in Figure 4.2(c) in hybrid format
5. Encode IP Address 10.20.30.40 in TLV format.
6. Refer to RFC 1213 for the following exercise:
   (a) Write the ASN.1 specifications for *sysServices.*
   (b) Illustrate the specifications with values for a bridge.
   (c) Illustrate the specifications with values for a router.
7. Write the object DESCRIPTOR and syntax of the following SNMP managed entities:
   (a) IP address 125.52.66.24
   (b) A row in the Interfaces table (the row specifications only, not the objects in the row)
   (c) MAC address of an interface card
8. In Exercise 4 of Chapter 1, you measured the percent packet loss using Ping tool, which depends on the ICMP group. Name the MIB objects that are used in the procedure and present the macros for the OBJECT TYPE.
9. Explain how you would determine whether a device is acting as a host or as a router using an SNMP command.
10. Refer to the IP Address Translation table shown in Figure 4.34 and Table 4.10, as well as the numbering convention shown in Figure 4.22 to answer the following questions:
    (a) List the columnar objects under *ipNetToMediaEntry.*
    (b) Draw an object instance table for *ipNetToMediaTable* as in Figure 4.23(b) without the row column. Fill three rows of data using MIB specifications.
    (c) Redraw the table in (b), now filling each cell in the table with an object instance identifier. Use *N* = 1.3.6.1.2.1.4.22.1 for *ipNetToMediaEntry* in the table.
11. You own a specialty company, ABC (Atlanta Braves Company), which sells hats and jackets. You obtained an OBJECT IDENTIFIER 5000 under *enterprises* node from IANA. You have two branch locations. Each has an inventory system that can be accessed by the IP address; which have the following OBJECT DESCRIPTORS:

    branch1 - 100.100.100.15
    branch2 - 100.100.100.16

    Each branch has two types of products whose inventory are

    hats
    jackets

    Hats are all of the same size and the inventory is a scalar value, hatQuantity.
    Jackets come in different sizes and the inventory is maintained in a table, jacketTable, whose columnar objects are

jacketSize (index)
jacketQuantity

Create a MIB module for your company. The objective is to find the inventory of any specific product while sitting in your office as president of the company.

(a) Draw a MIB subtree.
(b) Write a MIB module.

12. A network manager discovers that a network component is performing poorly and issues an order to the technician to replace it. Which MIB group contains this information for the technician to find out the physical location of the component?
13. How would you use one of the standard MIB objects to determine which one of the stations in a LAN is functioning as a bridge to the external network?
14. TCP is a connection-oriented protocol and UDP is a connectionless protocol. Identify differences in the two MIBs that exemplify this difference.
15. What OBJECT TYPE would you use to identify the address of the neighboring gateway from your local gateway?
16. An IT manager gets complaints from users that there is excessive delay in response over the Ethernet LAN. The manager suspects that the cause of the problem is excessive collisions on the LAN. She gathers statistics on collisions using the dot3StatsTable and localizes the problem to a single faulty network interface card. Explain how she localized the problem. You may use RFC 2358 to answer this exercise.
17. FDDI is heavily used as a backbone network in a corporate complex.
    (a) Draw a MIB tree for FDDI MIB. Limit your tree to the top five groups.
    (b) Develop a three-column table presenting entity, OID, and brief descriptions of the groups and tables under each group.
18. Two new managed objects, *ifName* and *ifAlias* were introduced in *ifMIB* module. Explain the purpose of these new managed objects in network management and give an example for each case.
19. Illustrate (a) the PPP over HDCL and (b) the cable access link with one downstream and two upstream channels using the interface sublayers shown in Figure 4.29.